Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Sistema Electrónico Textil para Señalización Ciclista mediante Reconocimiento de Gestos

Autor: Ángel Arlucea Almeida
Tutor: Antonio Luque Estepa

Dep. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Sistema Electrónico Textil para Señalización Ciclista mediante Reconocimiento de Gestos

Autor:
Ángel Arlucea Almeida

Tutor:
Antonio Luque Estepa
Profesor titular

Dep. Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Trabajo Final de Grado: Sistema Electrónico Textil para Señalización Ciclista mediante Reconocimiento de Gestos

Autor: Ángel Arlucea Almeida

Tutor: Antonio Luque Estepa

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

Agradecimientos

Este trabajo no hubiera sido posible sin la enorme disposición, paciencia y dedicación de Antonio Luque. Ha sido un auténtico privilegio conocerte y haber tenido la oportunidad de trabajar contigo, tanto en este proyecto como a lo largo de la carrera.

También quiero acordarme especialmente de los demás profesores del Departamento de Ingeniería Electrónica. Habéis hecho que no pueda estar más contento de haber tomado la decisión de escoger la especialidad de Sistemas Electrónicos.

Gracias a mi familia por estar ahí cuando las cosas no salían bien, especialmente a mi madre por su esfuerzo para hacer que esto fuera posible, por andar el camino conmigo, por señalarme Ingeniería de Telecomunicaciones en aquella interminable lista de opciones universitarias tras la PAU. Gracias a ti, Sara, porque con tu impulso e incondicional apoyo ha sido mucho más fácil culminar esta etapa de mi vida.

Imposible es, también, olvidarme de mis amigos y compañeros de alegrías y sufrimientos. Sin vosotros, mi paso por la ETSI se hubiera hecho mucho más cuesta arriba.

De todos y cada uno de mis profesores a lo largo de esta etapa de mi vida hay algo en este trabajo. Gracias a todos por enseñarme que sabía muy poco.

Ángel A. Almeida

Sevilla, 2016

Resumen

Debido al bajo coste de fabricación y venta de placas de desarrollo como Arduino o Raspberry Pi, se ha producido una expansión multidisciplinar a gran escala de proyectos basados en estas plataformas hardware.

En la actualidad comienza a popularizarse la incorporación de estos dispositivos en lugares donde anteriormente era impensable colocar aparatos electrónicos, tales como calzado, gafas, lámparas o contenedores de residuos.

En el campo textil también se ha aplicado esta revolución tecnológica. Son los dispositivos denominados 'wearables'.

Este Trabajo Final de Grado encaja en esta categoría, y su objetivo es diseñar y prototipar un sistema electrónico textil de señalización para ciclistas. Su funcionamiento se basa en el reconocimiento de una serie de gestos captados mediante un acelerómetro.

El sistema se compone de una placa de desarrollo especialmente diseñada para ser cosida a la ropa de forma sencilla y un acelerómetro conectado a la misma y situado en la muñeca, de forma que capte las aceleraciones producidas al mover el brazo y sea capaz, mediante un algoritmo, de reconocer el gesto realizado.

El objetivo es actuar sobre una serie de LEDs (diodos emisores de luz), encargados de señalizar la dirección hacia la que se desea girar en función del movimiento realizado con el brazo.

Abstract

Due to the low manufacturing and sale cost of development boards such as Arduino or Raspberry Pi, a high scale multidisciplinary expansion of projects based on these hardware platforms has taken place.

Nowadays the inclusion of these devices in places where previously was unthinkable is becoming popular, like in footwear, glasses, lamps or in rubbish bins.

A recent field of application of these devices is the textile one. The result of the mixture between technology and clothes is called “wearables”.

This project fits on this category. The target is to design and prototype a signaling textile electronic system for cycling. Its operation is based on the reconnaissance of gestures, captured using an accelerometer.

The system is composed by a microcontroller especially designed to be easily sewn to the clothes, and an accelerometer placed in the wrist and connected to it, capturing the movements of the arm. The system, by using an algorithm, will be able to recognize the gestures and act over the LEDs (light-emitting diodes) that form the signaling system.

These indicators will point the direction towards the cyclist wants to turn, according to the movement of his arm.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Gracias al avance tecnológico, la incorporación de dispositivos electrónicos en lugares donde antes era impensable hacerlo es cada día más habitual. El objetivo, como el de todo producto tecnológico comercial, es mejorar la calidad de vida del usuario y aumentar el confort durante la realización de ciertas actividades, así como incrementar la seguridad.

Los 'wearables' o dispositivos electrónicos aplicados en productos textiles pretenden mejorar el conocimiento y el control que el usuario tiene de su cuerpo mediante dispositivos biométricos, aumentar la cantidad de información a la que se tiene acceso de forma instantántea y sencilla mediante smartwaches o simplemente aportar algo distinto desde un punto de vista estético, usando LEDs y pantallas.

Uno de los ámbitos en el que se están comenzando a incorporar estos dispositivos, es el vial. En ocasiones, es complicado, desde el punto de vista del conductor de un vehículo a motor, interpretar los movimientos de un ciclista. Sus intermitentes son sus brazos y no siempre se utilizan de forma correcta o se ven con la suficiente antelación, por lo que la alerta al resto de conductores por un cambio de dirección es poco distinguible o, directamente, inexistente.

En España, tradicionalmente, el uso de la bicicleta ha sido residual en zonas urbanas. Sin embargo, con el fomento en los últimos años de su uso para desplazamientos dentro de la ciudad, el número de usuarios de este tipo de vehículos se ha disparado. Eso provoca que, al convivir un mayor número de bicicletas con los vehículos a motor, éstas necesiten hacer más visible aún los movimientos que van a realizar con el fin de evitar accidentes.

Según la Dirección General de Tráfico en su informe de Siniestralidad Vial de 2013 [1], el 73% de los accidentes de bicicletas se produjeron en vías urbanas. En el informe de Siniestralidad Urbana de ese mismo año [2], se refleja que el 75% de los accidentes de bicicletas implicaba a otro vehículo. En días laborables ese porcentaje aumentaba al 76% mientras que en fin de semana caía al 68%.

Lo más destacable de estos informes son los siguientes datos: entre las 8:00 y las 23:59h, el 74% de los accidentes (2989 siniestros) implicaban otro vehículo, pero entre las 00:00 y las 7:59, ese pocentaje aumentaba al 77% (158 siniestros) a pesar de caer significativamente el número de vehículos en circulación, lo que podría achacarse a las condiciones de baja luminosidad.

Esa sospecha se confirma si se observan directamente los datos estadísticos de 2013 de la Dirección General de Tráfico [3], reflejados en la siguiente tabla.

Tabla 0-1. Número de accidentes respecto a condiciones de luminosidad

Luminosidad	Tipo de vía	
	Urbanas	Resto
Crepúsculo	112	37
Noche: iluminación insuficiente	22	15
Noche: iluminación suficiente	309	17
Noche: sin iluminación	4	9

Como se puede ver en la tabla anterior, tanto durante el crepúsculo como por la noche en condiciones de iluminación suficente e insuficiente, la mayoría de accidentes sucedieron en vías urbanas.

En condiciones de falta de iluminación, esta estadística se invierte debido a la naturaleza del resto de vías (autopistas, autovías e interurbanas), ya que apenas existen vías urbanas no iluminadas.

1.1 Objetivo del Proyecto

El objetivo de este proyecto es diseñar y prototipar un dispositivo "wearable" de señalización de giro para su uso por parte de ciclistas, mejorando su visibilidad a la hora de realizar maniobras de cambio de sentido de la marcha.

El dispositivo se va a componer de una camiseta a la que se le incorporará una placa de desarrollo especialmente diseñada para ser utilizada en proyectos de electrónica textil, un acelerómetro que recoja los datos del movimiento del brazo izquierdo, que será el escogido para señalizar los movimientos de indicación de giro, y un hilo conductor que hará las veces de cable, uniendo el acelerómetro y la placa de desarrollo a través de la camiseta.

A dicho dispositivo se le incorporará un software diseñado específicamente con el propósito de detectar los gestos de señalización de giro indicados en la Guía Para Usuarios de la Bicicleta 2016 de la DGT [4]:

Figura 0-1. Señalización de giro a la izquierda

Figura 0-2. Señalización de giro a la derecha

Una vez detectado, se actuará sobre una serie de LEDs que conformarán el indicador de sentido del giro.

1.2 Estructura del documento

El documento se va a estructurar acorde a la realización del Proyecto.

En primer lugar, se encuentra la introducción, donde se encuadra la motivación del proyecto y la propuesta de trabajo.

Antes de comenzar a describir el proyecto, en el siguiente apartado se realizará una recopilación de los tipos de acelerómetros, de placas de desarrollo y las diferentes alternativas que existen en el mercado, tecnologías en desarrollo en el ámbito de los wearables, estudios realizados para medir el movimiento humano. Todo esto compondrá el Estado del Arte.

En el tercer punto se describirá el proyecto de manera general, haciendo hincapié en la estructura del mismo. Se verá una explicación mediante diagrama de bloques.

En el cuarto punto se introducirán los componentes que conforman el sistema de forma más detallada, explicando sus características y aspectos que los hacen interesantes para este proyecto.

Posteriormente, en el punto cinco, se detallará el algoritmo que se ha creado para el reconocimiento de los gestos del ciclista, así como se analizarán sus partes más importantes. El total del código se incorporará en los Anexos.

El siguiente punto a tratar es el proceso de pruebas del dispositivo, analizando con MATLAB® los datos que recoge el acelerómetro y realizando una serie de cálculos con ellos. Se describirá el proceso de montaje del prototipo paso por paso.

Finalmente se llegará a una serie de conclusiones y propuestas de mejora del dispositivo.

2 ESTADO DEL ARTE

Se inicia este proyecto haciendo una revisión de lo que se ha desarrollado en el ámbito de los "wearables", tanto en el ámbito comercial como en el experimental, así como en términos de seguridad vial e investigación en medición de la actividad y movimiento humano.

En la actualidad existen una serie de dispositivos que tratan de mejorar la seguridad de los ciclistas para prevenir accidentes. El objetivo suele ser hacer visible el comportamiento del ciclista al resto de vehículos. Para ello se ha aplicado y adaptado tecnología existente con el propósito de hacerla portable, ligera y resistente.

A la hora de medir el movimiento de una de las partes del cuerpo humano, un componente que no suele faltar en los dispositivos son los acelerómetros.

Normalmente, como se refleja en [5], las frecuencias y amplitudes de las aceleraciones que produce el cuerpo humano son bajas (a la hora de caminar, entre 0.8 y 5Hz en el tronco y aproximadamente 60Hz en los pies, donde se producen las mayores aceleraciones) Son un poco más altas cuando se realiza actividades físicas tales como saltos, caídas, sprints… Aun así, el 99% de las aceleraciones a la hora de caminar, se concentra en una frecuencia por debajo de 15Hz. El resto son consecuencia del impacto entre el calzado y la superficie por la que se camina.

Algo a tener en cuenta es que, en el movimiento humano, la frecuencia de las aceleraciones verticales es mayor que la de las horizontales. Desde un punto de vista frecuencial, la frecuencia de los movimientos de la cabeza es alta.

Si se analiza la amplitud de dichas aceleraciones, son mayores también en sentido vertical. En la parte superior del cuerpo suelen concentrarse entre -0.3 y 0.8g, mientras que en sentido horizontal se encuentran en el rango de -0.2 y 0.2g. Si se miden estas aceleraciones desde, por ejemplo, un tobillo, a la hora de andar se dan unas aceleraciones de entre -1.7 y 3.3g, mientras que en sentido horizontal son de entre -2.1 y 2.3g.

Independientemente de la técnica de medición de estas aceleraciones, una solución de compromiso a la hora de diseñar o escoger un acelerómetro para la medida del movimiento humano podría ser la de un dispositivo que sea capaz de detectar aceleraciones en el rango de -12 a +12g, y que pueda registrar frecuencias superiores a 20Hz. En general, en la cintura basta con un acelerómetro que detecte aceleraciones entre -6 y +6g.

Lo más allá de la sensibilidad, que dependerá de la aplicación, lo ideal es que este dispositivo sea capaz de detectar aceleraciones en las tres direcciones del espacio.

2.1 Acelerómetros.

Los acelerómetros [5] [6] [7] son dispositivos que convierten la aceleración a la que se encuentran sometidos,

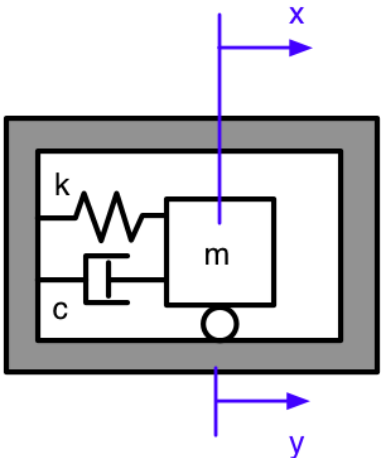


Figura 2-1. Acelerómetro mecánico.

en una señal eléctrica analógica proporcional a dicha aceleración. De forma conceptual, consiste en una masa conocida suspendida de un muelle. Al experimentar una aceleración, el muelle se elonga en función de la aceleración recibida, pudiendo medir el desplazamiento que sufre la masa y, calculando gracias a ello, la aceleración.

Existen tres tipos de fuerzas actuando sobre la masa: la externa, que es la que acelera la masa, la debida al *damping* o amortiguamiento, proporcional a la velocidad, y la que ejerce el muelle, que es proporcional a la posición. En equilibrio, la externa y la ejercida por el muelle son iguales.

La relación entre todos estos elementos, aplicando la segunda ley de Newton, es la siguiente:

$$F = ma = m\frac{\partial^2 x}{\partial^2 t} + c\frac{\partial x}{\partial t} + kx$$

siendo m la masa, a la aceleración y k la constante del muelle. El segundo término, en condiciones normales y lejos de la frecuencia de resonancia, puede obviarse, aunque no es el caso en acelerómetros de precisión que trabajen en un rango de frecuencias cercano a la de resonancia.

Hay tres métodos para convertir el desplazamiento de la masa en una señal eléctrica: midiendo el cambio de la resistencia de un material piezoresistivo, midiendo el cambio del valor de la capacidad entre elementos eléctricos fijos y móviles, y midiendo el cambio en la diferencia de potencial generada por un material piezoeléctrico.

2.1.1 Acelerómetros piezorresistivos

Estos acelerómetros se basan en la inclusión de un material piezorresistivo unido a la masa inercial. Al producir una aceleración, la masa ejerce una fuerza sobre el material, deformándolo y cambiando su resistencia. Están conectados a un puente de Wheatstone para producir una tensión eléctrica proporcional a la amplitud y la frecuencia de la aceleración de la masa del sensor. Es el mismo principio de funcionamiento de las galgas extensométricas.

Son más pequeños que los acelerómetros piezoeléctricos, pero requieren de una fuente de alimentación externa.

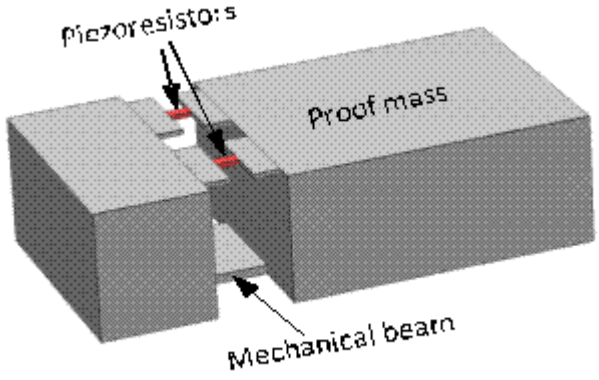


Figura 2-2. Acelerómetro piezorresistivo [8]

2.1.2 Acelerómetros piezoeléctricos

Los acelerómetros piezoeléctricos utilizan materiales piezoeléctricos como cristales (cuarzo) o cerámica. Consiste en un elemento piezoeléctrico pegado a una masa que, cuando ésta se mueve debido a una aceleración, realiza presión sobre el material piezoeléctrico, generando una diferencia de potencial.

Algo que destacar de este tipo de acelerómetros es que no responden a una aceleración constante.

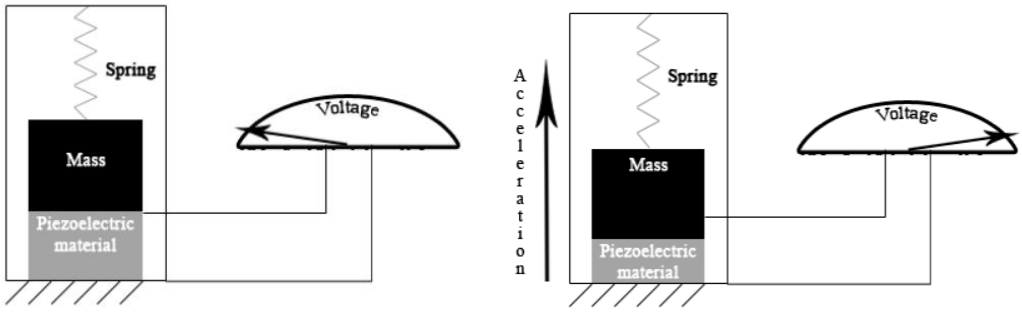


Figura 2-3. Acelerómetro piezoeléctrico. Principio de funcionamiento

2.1.3 Acelerómetros capacitivos

Su principio de funcionamiento se basa en la variación de la distancia de una serie de electrodos móviles respecto a unos electrodos fijos.

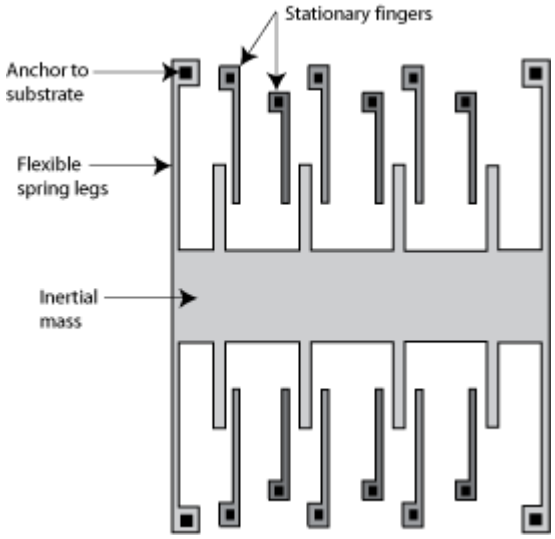


Figura 2-4. Acelerómetro capacitivo. Estructura

Estos electrodos móviles son parte del cuerpo móvil del acelerómetro, anclado en dos o más puntos al sustrato sobre el que se construyen los electrodos fijos.

La distancia entre los electrodos, en reposo absoluto, es constante. En caso de sufrir algún tipo de aceleración, los electrodos móviles (y, por ende, la masa o cuerpo móvil) se mueven respecto a los fijos.

La estructura electrodo móvil – espacio – electrodo fijo es un condensador. La capacidad de éste es constante en reposo, pero varía con el movimiento de la masa inercial de forma inversa a la distancia entre los electrodos.

Disponen de una alta sensibilidad y son muy lineales. Por su gran versatilidad y resistencia, suelen estar integrados en la amplia mayoría de dispositivos comerciales.

2.2 Placas de desarrollo ‘wearables’

Con la expansión de la electrónica textil, hay una serie de dispositivos que se han desarrollado con el fin de facilitar el conexionado y la integración en los tejidos. Paralelamente, se ha diseñado una serie de periféricos (para conectividad inalámbrica, acelerómetros y otro tipo de sensores, LEDs…) que conforman un auténtico ecosistema, simplificando mucho la incorporación de estos periféricos a cualquier proyecto ‘wearable’.

2.2.1 Arduino LilyPad

Existen cuatro tipos de LilyPad [9] en el mercado.

2.2.1.1 LilyPad Main Board

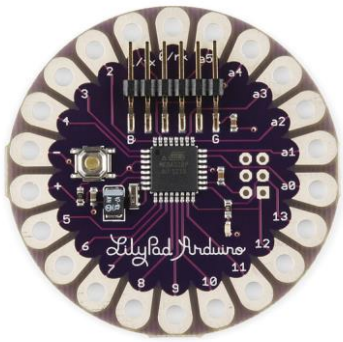


Figura 2-5. Arduino LilyPad Main Board

LilyPad Main Board utiliza un ATmega328V de 8 bits, dispone 1KB de memoria EEPROM, 2KB de SRAM y de 32KB de memoria Flash, operando a una frecuencia máxima de 20MHz. La resolución de su Convertidor Analógico Digital (abreviado como CAD o ADC) es de 10 bits.

Dispone de 22 pines en forma de anillo en el perímetro externo del PCB. Están perforados con el fin de coser la placa a la ropa con hilo conductor (se verá posteriormente), de forma que sea el propio hilo el que hace las veces de pista, conectando LilyPad a los distintos sensores que se deseen incorporar al proyecto.

Tiene 5cm de diámetro y un grosor de 0.8mm.

2.2.1.2 LilyPad Simple Board

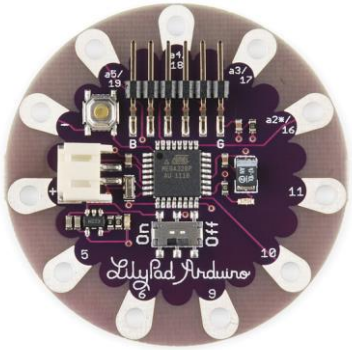


Figura 2-6. Arduino LilyPad Simple Board

Al igual que en el caso anterior, dispone de pines en el perímetro externo de su PCB. La diferencia es que, en lugar de 22, sólo dispone de 11 pines.

Dispone del mismo microcontrolador que la Main Board y sus características son iguales en todos los aspectos.

2.2.1.3 LilyPad Simple Snap

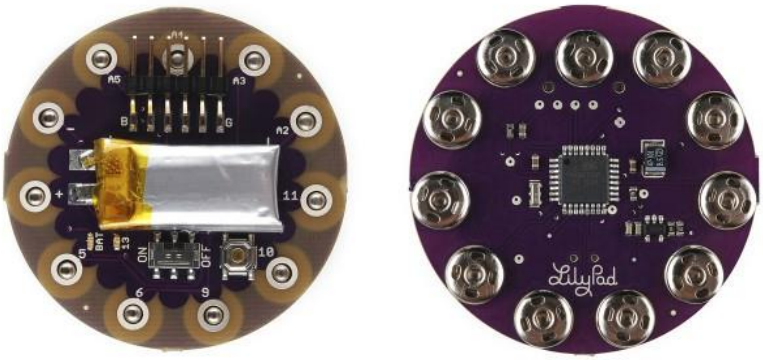


Figura 2-7. Figura 2-8. Arduino LilyPad Simple Snap

La diferencia de esta placa respecto a las anteriores reside en la forma de colocarla en la tela.

Dispone, al igual que la Simple Board, de 11 pines, pero no se cosen a la ropa, sino que son corchetes textiles. Se cose una parte del corchete a la ropa y sólo hay que presionar contra el pin de LilyPad Simple Snap para que encaje.

2.2.1.4 LilyPad USB

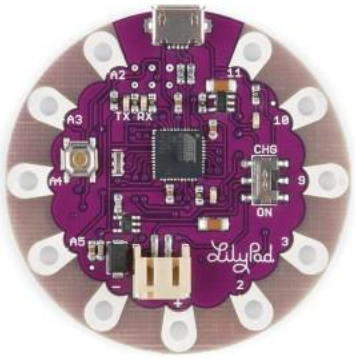


Figura 2-9. Figura 2-10. Arduino LilyPad USB

Dispone de 11 pines y, la principal diferencia respecto a Simple Board y Simple Snap es que el microcontrolador es un ATmega32U4.

Este micro tiene una frecuencia máxima de 16MHz (en lugar de los 20MHz que tenía el ATmega328V), pero aumenta el tamaño de su SRAM a 2.5KB (por los 2KB del 328V).

2.2.2 Adafruit

Adafruit [10] también ha desarrollado una serie de dispositivos ‘wearables’ siguiendo el concepto de LilyPad.

2.2.2.1 FLORA

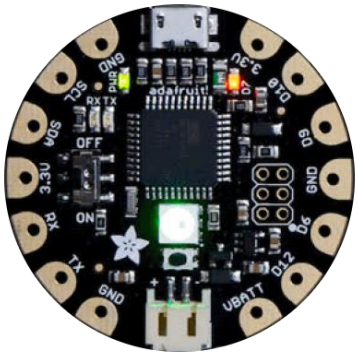


Figura 2-11. Adafruit FLORA

FLORA es la más potente de las placas de desarrollo ‘wearables’ que tiene Adafruit.

Dispone, al igual que LilyPad USB, de un ATmega32U4. La diferencia es que tiene 14 pines en lugar de 11 y que su tamaño es más reducido, pasando de los 5cm de diámetro que tenía la anterior a los 4.4cm que mide FLORA.

2.2.2.2 GEMMA

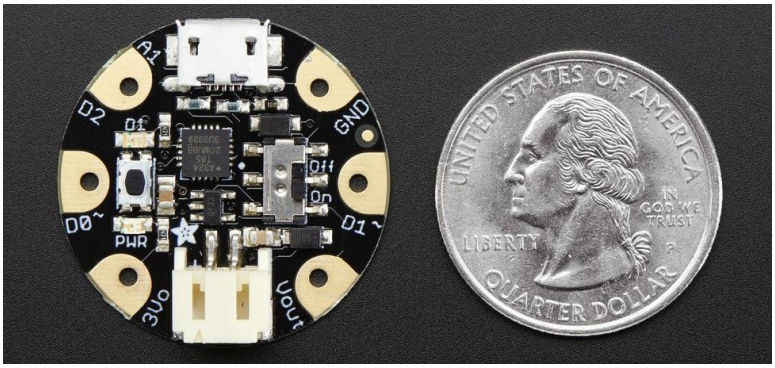


Figura 2-12. Adafruit GEMMA

Equipada con un ATtiny85, dispone solamente de 8KB de Flash, 0.5KB de SRAM y 512 Bytes de EEPROM.

GEMMA dispone 6 pines, por los 14 de FLORA. Con un diámetro de 2.8cm, es la más pequeña de todas.

A pesar de que es programable mediante el IDE de Arduino, no es totalmente compatible con éste. No dispone de conectividad por puerto serie y la conexión a través de puertos USB 3.0 suele ser problemática.

2.2.3 KeKeSmart

Disponen de una placa de desarrollo llamada KeKePad [11] y de un ecosistema de dispositivos anexo a todo lo desarrollado para Arduino, ya que son 100% compatibles.

2.2.3.1 KeKePad

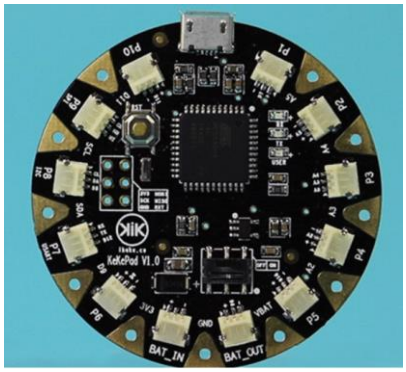


Figura 2-13. KeKeSmart KeKePad

De 5cm de diámetro y 1.2mm de espesor, dispone también de un ATmega32U4.

La diferencia de esta placa respecto a las anteriores, es que para su conectividad con los sensores se sirve de unos pequeños puertos de tres pines, donde se conectan cables muy finos y flexibles, de sólo 0.32mm diámetro.

Tiene 12 conectores de este tipo y 11 pines para coser con hilo conductor, con el que también es compatible.

2.2.4 Intel

En 2015, Intel presentó Curie [12], un módulo autónomo del tamaño de un botón que se puede incorporar en cualquier dispositivo o lugar.

2.2.4.1 Curie

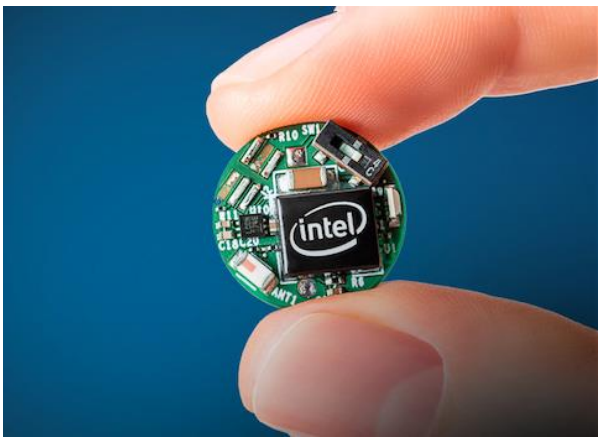


Figura 2-14. Intel Curie

Curie es un pequeño modulo con conectividad Bluetooth Low Energy (Bluetooth LE). Dispone de múltiples sensores, tales como acelerómetros y giróscopos, y también tiene tecnología de reconocimiento de patrones.

Dispone de una arquitectura de 32 bits, 384KB de Flash y 80KB de SRAM.

Para extraer la información de Curie, existe un entorno de desarrollo propio que facilita a los desarrolladores la gestión del dispositivo. La idea principal es utilizar Bluetooth para enviar la información al Smartphone y otros dispositivos electrónicos.

2.3 Mercado. Dispositivos existentes.

Para finalizar con el Estado del Arte, se van a mostrar una serie de dispositivos comerciales e ideas en vías de desarrollo. Existen varios dispositivos que se basan en proyectores y en visibilizar al ciclista por la noche, ya sea mediante luces, sprays o directamente proyectando una señal en el asfalto. Sin embargo, los que se van a analizar son los directamente relacionados con el objetivo del proyecto, es decir, con la señalización del sentido del giro, no con visibilizar la posición del ciclista.

2.3.1 Cyclee

Cyclee [13] es un proyector de señales en la espalda del ciclista. Estas señales, completamente configurables, avisan a los automovilistas de las intenciones del ciclista a la hora de girar, parar o circular. Al contrario que otras alternativas wearables, el proyector no está directamente colocado en la camiseta, sino que es un añadido al sillín. Esto posibilita la utilización de cualquier tipo de ropa como "lienzo", pudiendo cambiar las señales proyectadas desde una APP móvil.

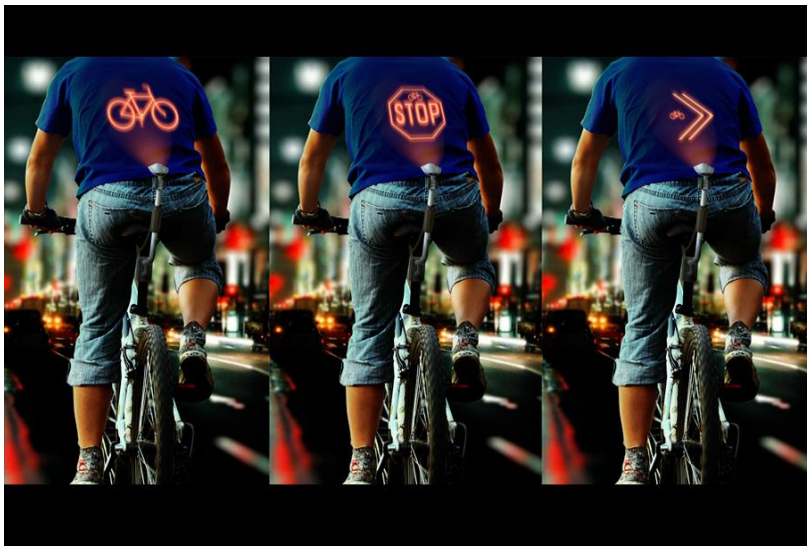


Figura 2-15. Cyclee

2.3.2 Zackees

Zackees [14] es una empresa que ha desarrollado unos guantes luminosos que se encienden utilizando los dedos.

Figura 2-16. Zackees

Estos guantes disponen de un sensor de luz ambiente para adaptar la potencia del panel LED de señalización a la luz ambiental.

El método de encendido y apagado es mediante unas pequeñas placas de metal que se encuentran entre los dedos pulgar e índice. Simplemente hay que juntar los dedos para encender los LEDs del guante y volver a juntarlos para apagarlos.

2.3.3 Safe Ride

Desarrollado por unos estudiantes de la facultad de ciencias de la UNAM [15], consiste en una serie de señales luminosas que se activan cuando ciertos sensores detectan movimiento en los hombros, indicando la dirección del giro. Está aún en fase de prototipo.

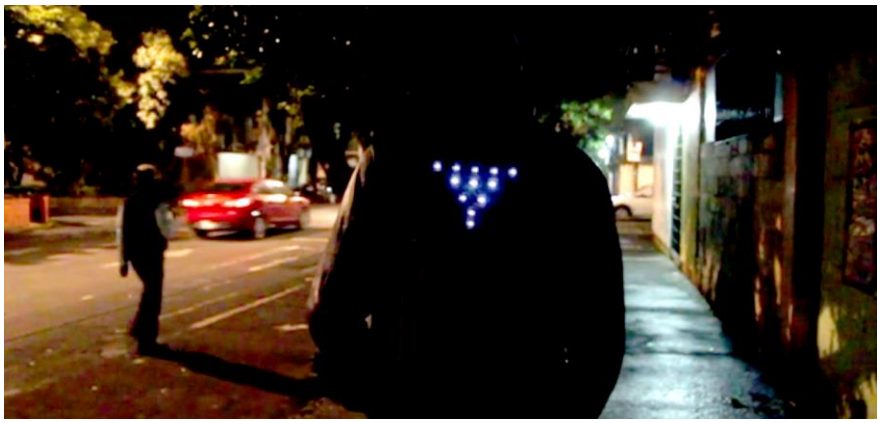


Figura 2-17. Safe Ride

2.3.4 Lumos

Lumos [16] ofrece un caso diferente a los anteriores, ya que la señalización no va en la ropa sino en el casco del ciclista.

Figura 2-18. Lumos

Dispone de intermitentes y de luz de freno. La señalización se produce mediante LEDs: naranjas para los intermitentes laterales, rojos para la luz de freno y blancos para la luz frontal del casco, que señaliza la posición.

El encendido y apagado de los LEDs se realiza mediante un dispositivo wireles situado en el manillar y que dispone de dos botones, uno por cada sentido de giro.

Figura 2-19. Lumos. Manillar

Cuando el casco detecta que el ciclista está decelerando, enciende en rojo todos los LEDs laterales y el trasero.

2.3.5 Blackpack

Blackpack [17] es un prototipo de mochila intermitente. Consiste en una mochila que dispone una banda de LEDs que se mueven indicando el sentido del giro del ciclista.

Figura 2-20. Blackpack

Aún es un concepto y su principio de funcionamiento no es conocido, por lo que no es posible determinar si requiere del algún pulsador para indicar el sentido del giro o si dispone de algún gadget añadido para detectarlo automáticamente.

La idea principal del autor es combinar la resistencia a los golpes, la humedad y la lluvia, con la seguridad.

2.4 Resumen y conclusiones

El número de dispositivos 'wearables' disponibles en el mercado han sufrido un gran aumento en los últimos años gracias al grado de desarrollo tecnológico que se ha alcanzado en acelerómetros, giróscopos y todo tipo de sensores de temperatura, humedad, sudoración, etc. La miniaturización e integración de estos sensores ha sido clave para su incorporación en tejidos sin afectar al confort.

Como consecuencia, a la hora de estudiar los movimientos humanos se ha logrado una precisión nunca antes alcanzada, y la combinación de estos sensores con placas de desarrollo 'wearables', tales como la familia LilyPad o Adafruit, ha facilitado la creación de todo tipo de dispositivos, entre los que se encuentran los destinados al ámbito de la seguridad vial.

En este campo, se han desarrollado una serie de productos, vistos en este apartado, que pretenden visibilizar al ciclista y alertar al resto de conductores de sus intenciones a la hora de cambiar el sentido de la marcha. Dispositivos comerciales como Cyclee, Zackees y Lumos, así como otros en desarrollo como Safe Ride y Blackpack, dan buena cuenta de ello.

En la mayoría de los dispositivos que se han recogido en este apartado, el funcionamiento se basa en pulsadores. A lo largo de este proyecto se verá una nueva alternativa al respecto, mediante la utilización de acelerómetros y un firmware de reconocimiento de gestos.

En el siguiente apartado se presenta el diseño del dispositivo, así como también se especifican las características que debe tener el dispositivo para que sea óptimo para este objetivo.

3 DISEÑO DEL SISTEMA

Para realizar este proyecto, hay que realizar un estudio de los requisitos necesarios para conseguir el objetivo de un sistema autónomo, rápido y preciso.

El sistema que se va a desarrollar en este proyecto se compone de tres dispositivos:

- Un acelerómetro, el cual se ubica en la muñeca y que recogerá los gestos realizados por el ciclista.
- Un microcontrolador, donde se envían los datos del acelerómetro y se procesan acorde a un algoritmo de reconocimiento de gestos. Estará situado en la parte delantera de la camiseta.
- Una serie de LEDs en la parte posterior de la camiseta que compondrán la señal luminosa de giro. Estarán conectados al microcontrolador y, en caso de realizar con el brazo algún gesto reconocible, se encenderán en función del gesto detectado.

Un sistema autónomo, con conexión a sensores externos y aislado del resto del ecosistema de dispositivos electrónicos, no puede requerir de agentes externos para el control, la gestión y la presentación de los datos.

Con este fin, se hace necesario contar con placas de desarrollo que tengan puertos de entrada/salida, que dispongan de comunicación serie para facilitar la depuración del sistema y que tengan memoria suficiente para poder trabajar autónomamente, no sólo leer los datos de los sensores integrados y enviarlos inmediatamente a un tercer dispositivo, donde se analizan y presentan.

Desde un punto de vista temporal, el sistema debe ser lo suficientemente rápido a la hora de:

- Realizar la medición en el sensor
- Enviar el dato a la placa de desarrollo.
- Preprocesar y almacenar.
- Analizar.
- Actuar consecuentemente sobre otros dispositivos.

Como se puede observar, son varias etapas, en las que entran en juego un gran número de partes del sistema, tales como sensores, convertidores analógico-digitales (en caso de que sea necesario convertir el dato porque el sensor sea analógico), memorias… A más dispositivos involucrados, más lento es el sistema.

En este caso, para medir un gesto, lo primero que hay que tener en cuenta para la elección de un sensor es la ubicación del mismo.

Para la medida de gestos en cualquier lugar del cuerpo la frecuencia debe ser superior a 20Hz, y la sensibilidad debe abarcar desde -12g hasta +12g [5]. Esto es lo ideal, es decir, es lo necesario para poder medir el gesto con una precisión rozando el 100%.

En este proyecto lo que se busca es la facilidad de distinguir un gesto de otro, pero no es necesario alcanzar niveles de precisión tan altos como los anteriores, por lo que habrá que encontrar un valor de compromiso para que el sensor no se sature. Para aceleraciones como las que se van a producir al realizar los gestos, que no van a ser bruscas, será suficiente con una sensibilidad de 1.5g.

Teniendo esto en cuenta para el acelerómetro, hay que decidir también la placa de desarrollo a utilizar.

Se pretende trabajar con lo que, a lo largo del proyecto, llamaremos patrones (gestos “modelo” o de referencia, idénticos a los de las figuras 1-1 y 1-2, con los que comparar los datos que se leen del acelerómetro instantáneamente) y con el gesto leído del acelerómetro, por lo que, dimensionando los vectores de datos de cada patrón y del gesto, se alcanzará un valor del tamaño de la memoria necesaria como especificación principal y más importante.

Teniendo en cuenta que realizar y mantener un gesto como los de las figuras 1-1 y 1-2 toma entre 2 y 3 segundos, por alcanzar un valor de compromiso adecuado para no cortar el gesto, se va a suponer que se realiza en 3.5 segundos. Teniendo en cuenta también que la frecuencia debe ser superior a 20Hz y suponiendo el doble de

frecuencia, 40Hz, se debería tomar una muestra cada 8ms. Para obtener más datos, como no tenemos ninguna limitación superior en frecuencia, se va a decidir bajar ese tiempo a 3ms, lo que quiere decir que en esos 3.5s se van a tomar 1166 muestras. Por poner un número redondo, se decide escoger los vectores de 1200 componentes, siendo el tiempo final de muestreo de 2,9ms y una frecuencia de aproximadamente 345 Hz, muy por encima de la necesaria.

Si se realizan los cálculos, 1200 datos para cada eje del acelerómetro (X, Y y Z) implican 3600 datos para el gesto y 7200 para los patrones (hay dos, el de girar a la derecha y el de girar a la izquierda), en total 10800 datos, lo cual no es nada adaptable a los dispositivos que se han descrito en el estudio del Estado del Arte. Habrá que dimensionar estos datos para poder trabajar con ellos en estos dispositivos, como se verá en la explicación del Apartado 5 dedicada al Código 1.

Así pues, se va a buscar un dispositivo con amplia memoria Flash y SRAM para poder acoger toda esta información. Como va a ser prácticamente imposible trabajar con esos 10800 datos en memoria y, sabiendo por tanto que va a ser necesario dimensionar el tamaño de los vectores de los datos, con entre 2 y 2.5 KB de SRAM y unos 30 KB de memoria Flash va a ser más que suficiente, ya que suponiendo que cada dato son 2 Bytes, entre Flash y RAM son 32500 Bytes, más que los 21600 Bytes que serían necesarios en caso de mantener los vectores a 1200 componentes cada uno (y no va a ser el caso).

Como especificación secundaria, la frecuencia de reloj debe ser suficiente para realizar cada uno de los procesos descritos arriba con la mayor premura posible. Igualmente, el CAD de la placa de desarrollo debe ser lo más rápido posible en el caso de escoger un acelerómetro analógico, de forma que no se pierda la principal ventaja de este tipo de sensores.

Para ello, un reloj de 8MHz como el que tiene FLORA resulta más que suficiente, como se verá a lo largo del proyecto.

Una vez descritas las especificaciones de los dispositivos que componen el proyecto, se va a proponer un diseño para el mismo.

El diagrama de bloques general es el siguiente:

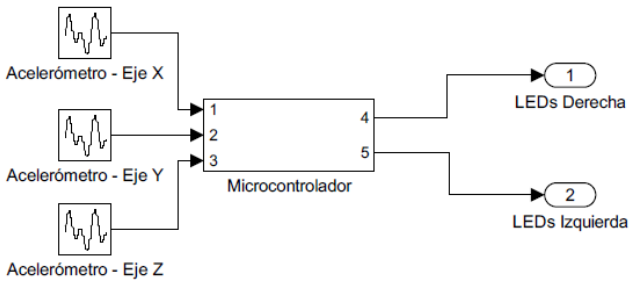


Figura 3-1. Esquema general del sistema.

El acelerómetro escogido, como se verá posteriormente, es analógico, por lo que será necesario convertir la señal que transmite en una señal digital para su procesado. Una vez se ha realizado esta conversión, es momento de detectar si la señal se corresponde a alguna señal conocida.

Estas señales conocidas o de referencia son los denominados “patrones”. Va a haber dos patrones de referencia, uno por cada sentido del giro. Estos patrones son conocidos en tiempo de compilación y se encuentran en la memoria de programa del micro.

De esta forma, contraponiendo la señal detectada al patrón, se podrá distinguir si el ciclista ha realizado el gesto de girar a la derecha o de girar a la izquierda. Si el gesto realizado por el ciclista no se detecta como válido debido a la falta de parecido con las señales patrón, se obvia y se sigue analizando la información que el acelerómetro sigue transmitiendo.

La correspondencia entre la señal que transmite el acelerómetro y el patrón de referencia se va a buscar mediante el uso de la correlación cruzada (se hará hincapié en ella en el punto 5, al explicar el código correspondiente a la identificación de gestos). Se compararán los tres ejes de cada gesto por separado con los correspondientes (X con X, Y con Y y Z con Z) de cada uno de los dos patrones y, si el parecido en los tres ejes es razonablemente alto respecto a uno de los dos patrones, decimos que existe coincidencia.

En ese caso, la búsqueda de similitud arroja un resultado positivo, es decir, el microcontrolador detecta que el gesto realizado se corresponde con el gesto patrón de giro a la derecha o a la izquierda (nunca va a ser parecido a los dos a la vez) y se actúa sobre los LEDs correspondientes a la indicación del sentido del giro que se ha detectado.

En la Figura 3-2 se ilustra todo esto.

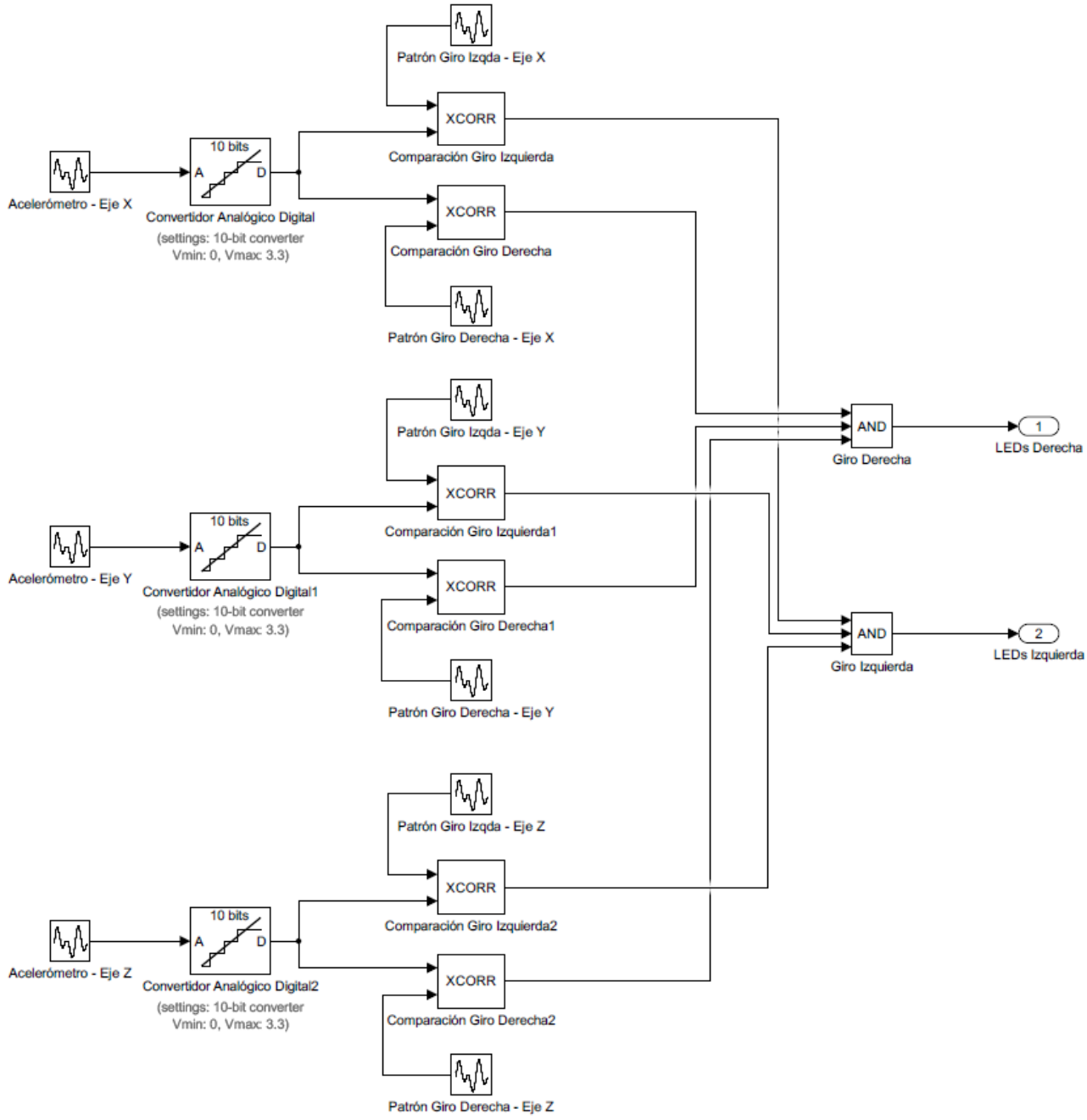


Figura 3-2. Funcionamiento interno del sistema.

Por hacer una analogía entre la Figura 3-1 y 3-2, visualizando esta última de izquierda a derecha, los bloques que van desde el CAD hasta el AND comparativo, forman parte del microcontrolador.

Tras decidir los requisitos del sistema, se van a describir los componentes elegidos para el sistema.

4 COMPONENTES DEL SISTEMA

En este apartado se va a hablar de los dispositivos que componen el sistema. Primero se van a presentar y a describir por separado y, posteriormente, se comentará el montaje del prototipo y sus especificaciones y características.

4.1 Adafruit FLORA

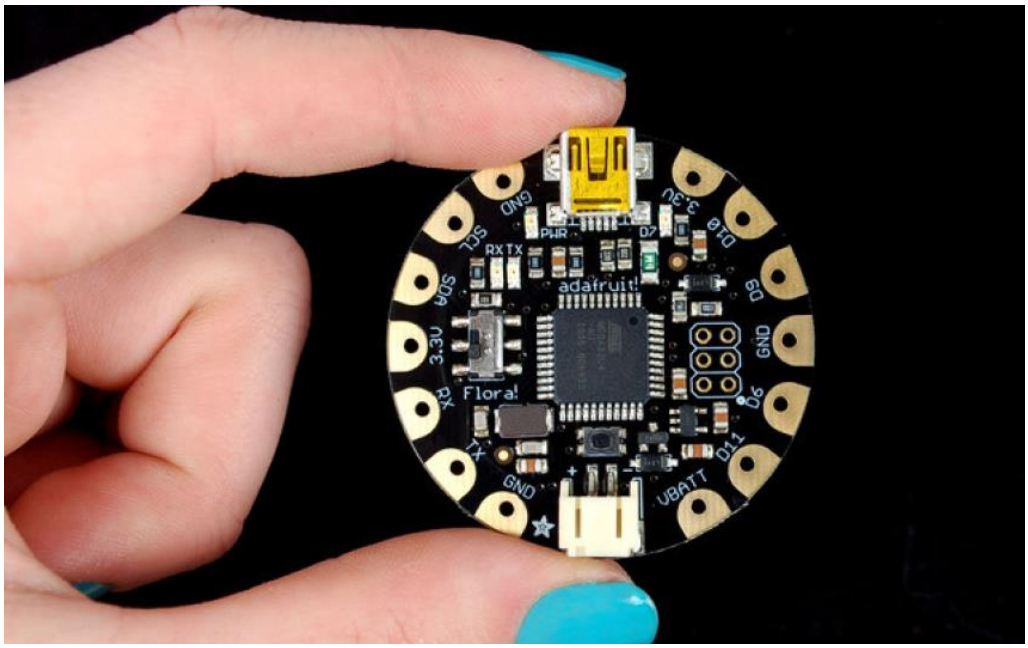


Figura 4-1. Adafruit FLORA

FLORA [18] ha sido la placa de desarrollo escogida para este proyecto. El motivo es que su diseño es específico para dispositivos wearables, tiene un precio muy competitivo y se adapta a la carga de trabajo que se espera tener en el microcontrolador. El PCB es circular, con los pines en la parte exterior, de forma que sea sencillo coserla a la ropa utilizando hilo conductor (se tratará posteriormente).

Sus dimensiones son muy reducidas, midiendo 4.445 centímetros de diámetro y pesando tan sólo 4.4 gramos, por lo que su incorporación a cualquier prenda de ropa no supone una molestia para la persona que la viste.

Dispone de un microcontrolador ATMEL ATmega32u4 de 8 bits [19] completamente compatible con Arduino, por lo que sólo es necesario incorporar sus drivers al IDE de Arduino para comenzar a trabajar con ella.

Tiene un CAD de 10 bits, lo que permite convertir señales analógicas en digitales a valores enteros entre 0 y 1023.

FLORA dispone de 32KB de memoria Flash, 1KB de memoria EEPROM y 2.5KB de SRAM.

Funciona a 8MHz y consume 8mA en reposo. Éste se incrementa a 20mA en periodo de actividad y, en caso de uso del LED integrado en el pin #D7, aumenta en 2mA más.

La flexibilidad existente a la hora de alimentar la placa hace que sea muy versátil. Además de alimentarse a través del puerto MicroUSB, dispone de un puerto JST de dos pines. Mediante él, es posible incorporar una batería de 3.5V hasta 9V (aunque no se recomienda una alimentación superior a 6V para evitar sobrecalentamiento). Dispone de un diodo schottky y de un regulador de tensión LN2985, lo que evita que, a pesar de alimentarlo a 9V, la placa se dañe físicamente y pueda presentar un mal funcionamiento, incluso si se

alimentase de forma inversa.

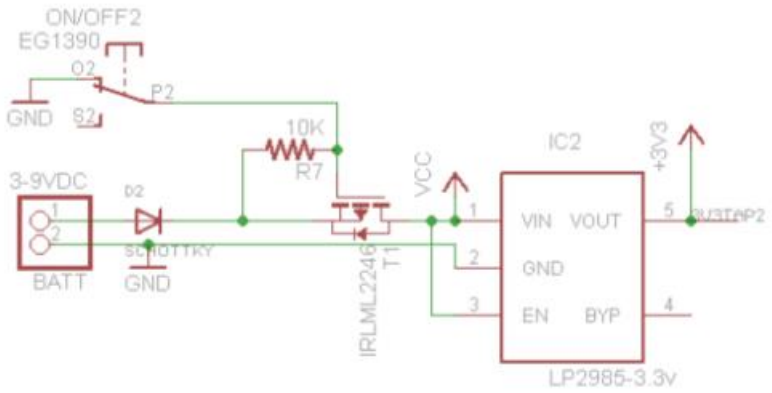


Figura 4-2. Alimentación de FLORA. Esquemático.

Utilizando ese puerto JST, es posible conectar múltiples tipos de baterías: Ion Litio, Litio-Ferrofosfato, alcalinas, recargables de Níquel-Metal hidruro o, las ya en desuso, Níquel-Cadmio.

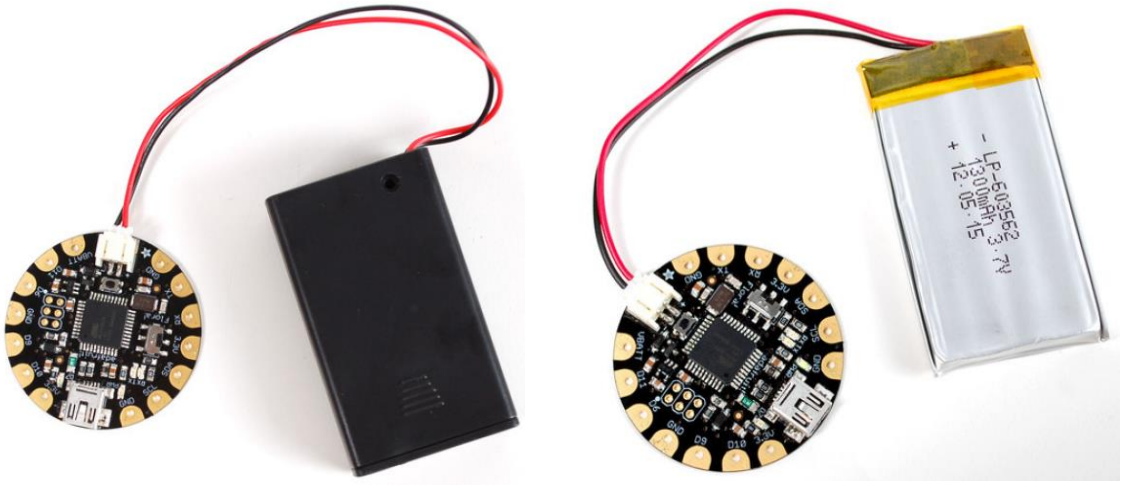


Figura 4-3. Alternativas de alimentación de FLORA

Algo a destacar de FLORA es que tiene un pin llamado VBATT. Este pin sirve para alimentar, directamente de la batería, aquellos componentes que requieran más de los 150mA que hay a la salida del regulador. Al ser un pin de alimentación, es de sólo salida.

Igualmente, la placa soporta hardware SPI.

El esquemático al completo y el pinout del dispositivo quedan recogidos en Anexo correspondiente

4.2 Acelerómetro MMA7260Q

Figura 4-4. MMA7260Q

El MMA7260Q [20] es un acelerómetro capacitivo analógico de tres ejes de bajo coste. Suele utilizarse como detector de caída libre, podómetro, estabilización de imagen en teléfonos, detector de inclinación…

Tiene un rango de sensibilidades seleccionables entre 1.5g y 6g. En funcionamiento, su consumo es bajo y su tensión de operación está en el rango de las que ofrece FLORA.

Su sensibilidad es alta e incorpora un filtro paso de baja para acondicionar la señal.

Esto, junto con una alta resistencia a impactos y su reducido tamaño, lo hacen perfecto para este proyecto.

Recopilamos sus características más importantes en la siguiente tabla.

Tabla 4-1. Características MMA7260Q

Característica	Valor
Sensibilidad	1.5g / 2g / 4g /6g
Consumo de Corriente	500µA
Tensión de Operación	2.2V – 3.6V
Dimensiones	6mm x 6mm x 1.45mm
Sensibilidad	800 mV/g @ 1.5g
Temperatura de Func.	-20º a +85º

Los valores máximos admitidos de aceleración, tensión y distancia de caída son:

Tabla 4-2. Valores máximos admitidos

	Valor
Aceleración Máxima	±2000g
Tensión de Entrada	-0.3 a +3.6V
Distancia de Caída	1.8m

Si observamos los valores típicos de funcionamiento, se ve que es precisamente lo que se busca: bajo consumo de corriente (muy por debajo de 150mA) y una tensión de 3.3V.

Tabla 4-3. Valores típicos de funcionamiento

Característica	Valores típicos
Tensión de func.	3.3V
Corriente de func.	500µA

Los pines de selección del rango de sensibilidad se llaman g-Select. Se van a seleccionar según esta tabla para ponerlos a 1.5g, valor más que suficiente para el objetivo de este proyecto.

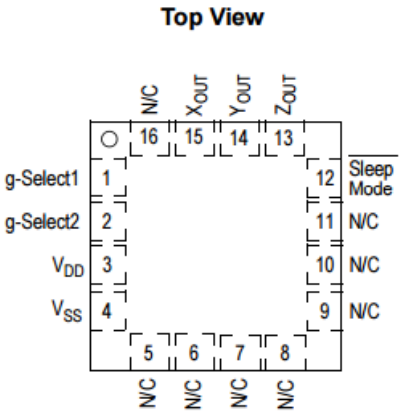


Figura 4-5. Pin Out MMA7260Q

Tabla 4-4. Rango y sensibilidad en función de pines g-Select

g-Select1	g-Select2	g-Range	Sensibilidad
0	0	1.5g	800mV/g
0	1	2g	600mV/g
1	0	4g	300mV/g
1	1	6g	200mV/g

Para el correcto funcionamiento del acelerómetro, el diagrama de conexión recomendado es el siguiente:

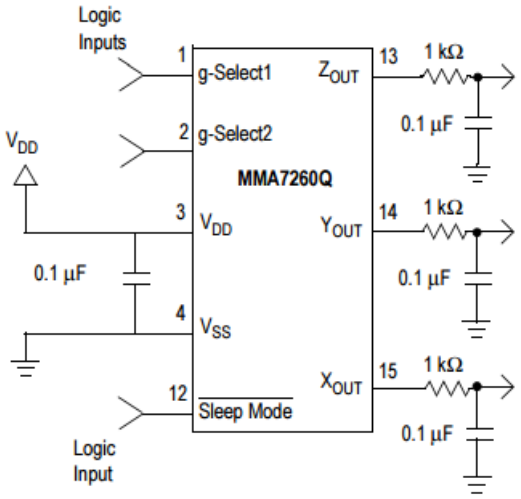


Figura 4-6. Diagrama de Conexión MMA7260Q

Se va a utilizar este diagrama para realizar el conexionado del acelerómetro.

El nombre de los pines y su descripción se puede encontrar en el Anexo.

Hay de tener en cuenta que el fabricante recomienda minimizar la distancia entre el acelerómetro y el microcontrolador donde vamos a recoger los datos. En este proyecto, como veremos durante la descripción del montaje y debido a la naturaleza de la aplicación a la que se destina el dispositivo, no se va a tener demasiado en cuenta esta recomendación por su baja influencia en el funcionamiento del mismo.

4.3 Hilo conductor

Figura 4-7. Hilo conductor de acero inoxidable

Al ser un Proyecto orientado a una aplicación textil, es necesario utilizar un conductor que permita unir la ropa con el dispositivo. Esa es la función del hilo conductor [21].

Es un fino hilo hecho de dos fibras de acero inoxidable 316L. Una de las características más notables de este acero es su alta resistencia a la corrosión [22]. Es importante este detalle porque va a hacer que la durabilidad del hilo y sus propiedades se mantengan a pesar de los lavados que se le puedan hacer al prototipo textil.

Figura 4-8. Detalle de las dos fibras que componen en hilo conductor

También su maleabilidad es un punto a favor, ya que hace que el hilo sea muy adaptable a las diversas formas que se necesiten darle a la hora de coserlo a la ropa.

Al tacto parece hilo común, aunque es ligeramente más rígido. Su grosor es de $0.2mm$ y presenta baja resistencia, sólo $0.5\Omega/cm$. Es ideal para aplicaciones wearables que trabajen por debajo de $50mA$, como la utilización de LEDs.

4.4 LEDs

Los LEDs van a ser los encargados de señalizar el sentido del giro del ciclista. Debido a la necesidad de ser visibles también durante el día, se ha escogido un modelo de 5mm de alta luminosidad.

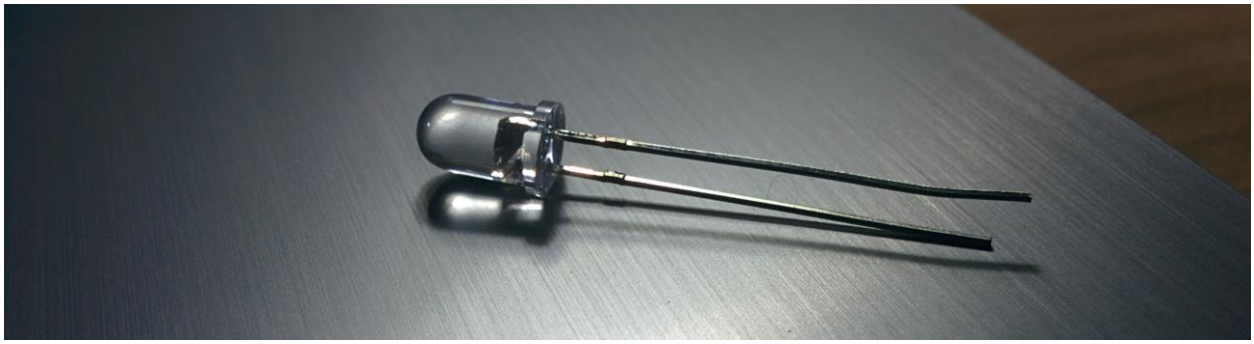


Figura 4-9. LED de alta luminosidad

Como se verá en el próximo apartado, las señales van a estar compuestas por 5 LEDs en paralelo cada una. El esquemático que se va a seguir para garantizar su correcto funcionamiento es el siguiente:

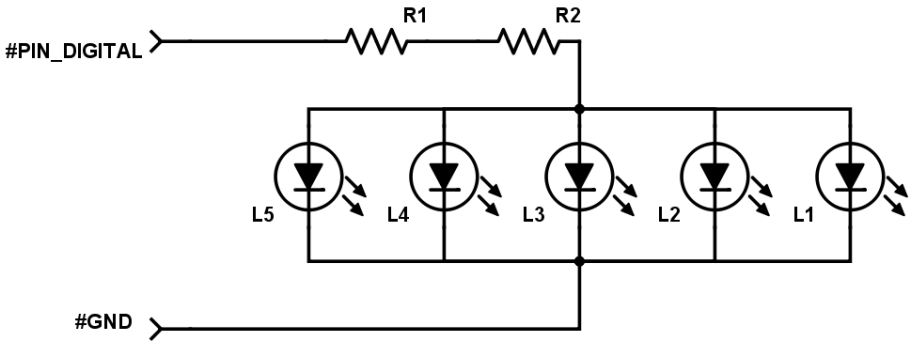


Figura 4-10. Conexionado de los LEDs. Esquemático.

Los LEDs van a ir en serie con dos resistencias de 82Ω. El #PIN_DIGITAL será el pin, asignado en FLORA y configurado como salida, que se activará al detectarse un gesto. La señal le llegará a los LEDs y se encenderán acorde a un número de repeticiones.

Al haber dos direcciones de giro, se construirán dos dispositivos diferentes siguiendo el esquemático anterior. Eso significa que en FLORA habrá asignados dos pines distintos como salida, uno para cada sentido del giro.

4.5 Prototipo

Una vez se conocen los componentes escogidos para el proyecto, es el momento de pasar a describir el montaje. Para llevar a la práctica el diseño del sistema propuesto, se va a seguir el siguiente concepto:

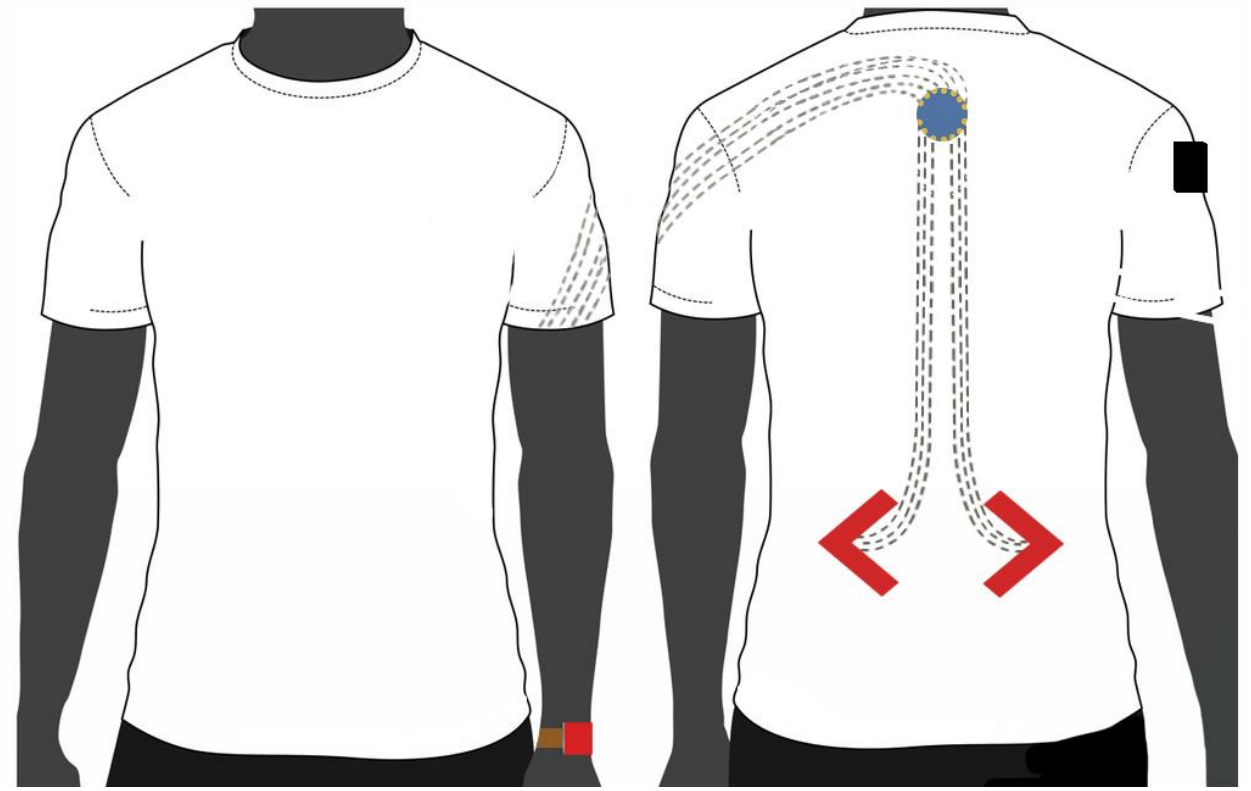


Figura 4-11. Concepto del sistema

En la figura anterior, se pueden ver cada una de las partes del sistema: la placa de desarrollo FLORA en el pecho, el hilo conductor, la batería en la manga derecha y el acelerómetro en la muñeca. No se observan en la figura las conexiones entre el acelerómetro (la pulsera roja situada en la muñeca) y FLORA. Debido a su condición de prototipo, las vamos a realizar utilizando cables cocodrilo-cocodrilo. Igualmente, se va a introducir el PCB del acelerómetro en una caja de Raspberry Pi adaptada para su utilización, ya que las dimensiones son compatibles.

En primer lugar, se mostrará parte del prototipo, compuesto por la placa de desarrollo FLORA y por el acelerómetro, sin incorporar los LEDs. Finalmente se incorporarán los LEDs y el soporte textil, dando la construcción del prototipo por finalizada. Las imágenes que se mostrarán van a ordenarse de forma lineal, desde el inicio de la construcción del prototipo hasta su finalización.

Durante la fase de pruebas se han utilizado cables cocodrilo-cocodrilo para facilitar el montaje y la depuración de los errores del prototipo. Estos cables no podían ser de diferente tipo debido al especial diseño de los pines de FLORA, planos y situados en el perímetro exterior de su PCB. La correspondencia de los colores con cada uno de los ejes es la siguiente:

Tabla 4-5. Acelerómetro. Correspondencia Color – Valor/Eje

Color	Valor/Eje
Rojo	VCC
Negro	GND
Verde	Eje X
Amarillo	Eje Y
Blanco	Eje Z

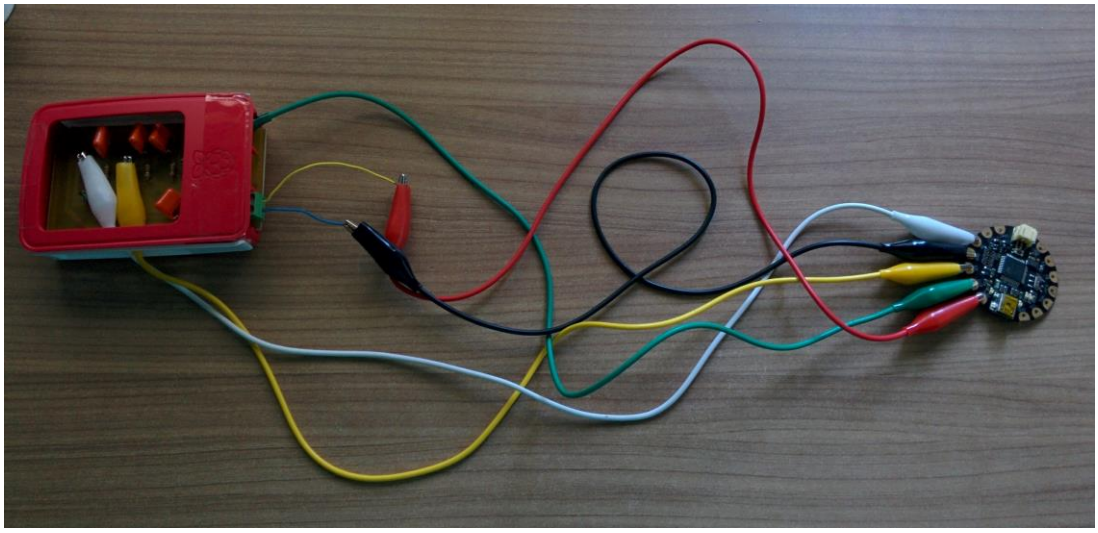


Figura 4-12. Conexión Acelerómetro – Adafruit FLORA. Vista general.

El acelerómetro se encuentra físicamente en la parte inferior del PCB. En la parte superior se pueden observar el resto de componentes de su montaje: condenadores, resistencias, pines de conexión a los tres ejes y jumpers para cambiar la sensibilidad del dispositivo.

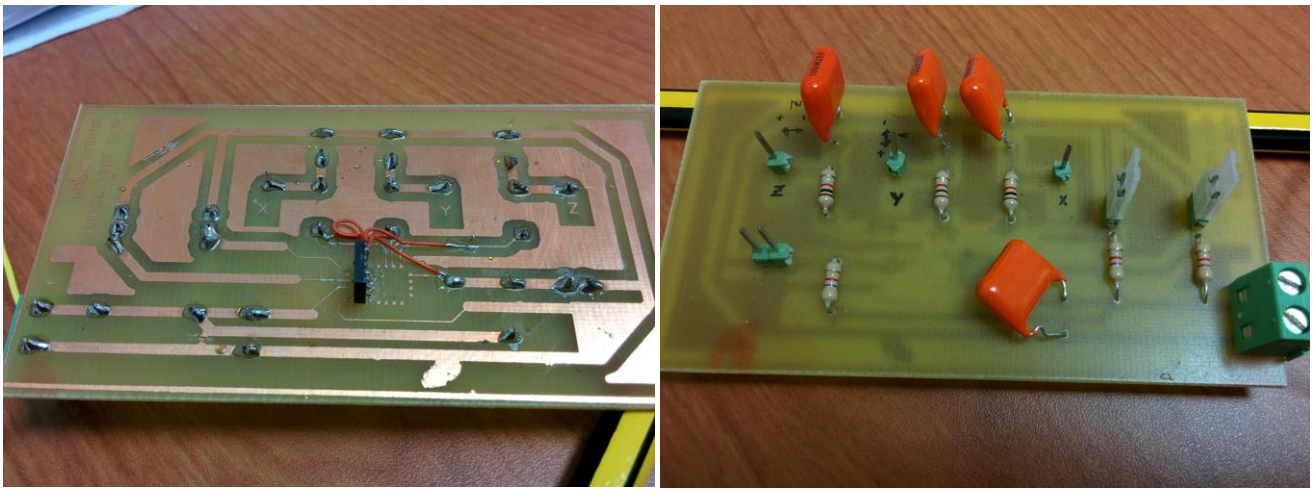
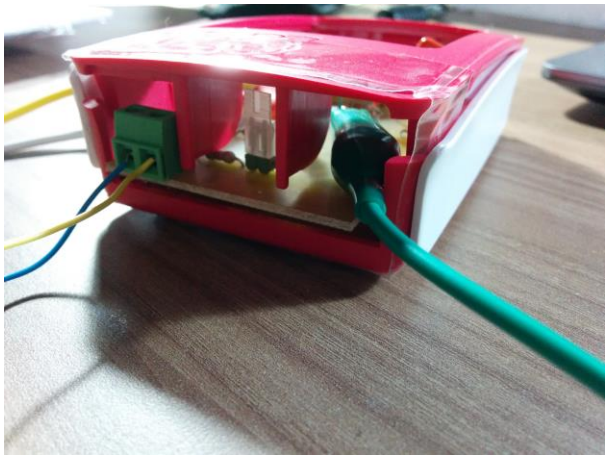


Figura 4-13. Acelerómetro MMA7260Q. PCB.

Se ilustran a continuación algunas características previamente comentadas del acelerómetro.



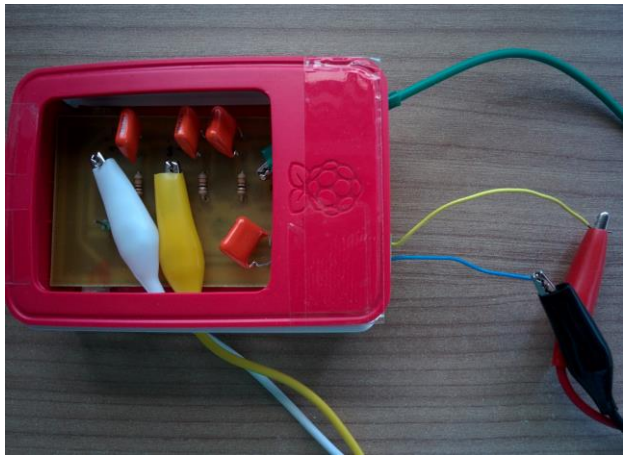


Figura 4-14. Acelerómetro. Alimentación y vista cenital.

Figura 4-16. Acelerómetro. Detalle de conexión de los ejes.

Figura 4-15. Acelerómetro. Detalle de gestión de cableado.

Como puede observarse, se han aprovechado los huecos de los puertos USB y HDMI de la Raspberry Pi para sacar los cables al exterior de forma ordenada, evitando así enredos entre ellos.

El hecho de que la caja sea completamente desmontable ha facilitado el trabajo.

Seguidamente se muestra el conexionado del acelerómetro con FLORA, así como la correspondencia Color – Valor/Eje – Pin.

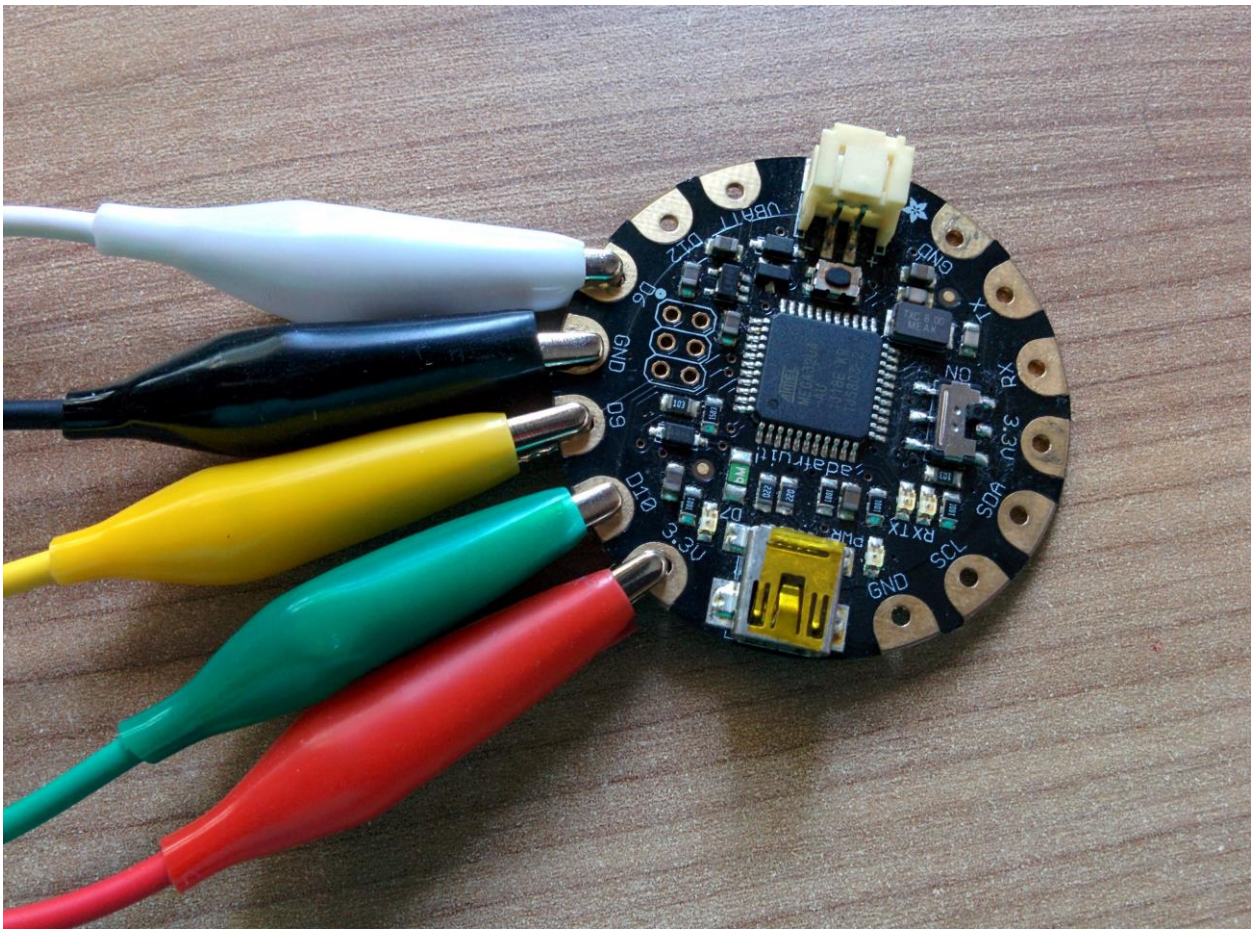


Figura 4-17. Adafruit FLORA. Conexionado con el acelerómetro.

Tabla 4-6. Correspondencia Color - Valor/Eje - Pin

Color	Valor/Eje	Pin
Rojo	VCC	3.3V
Negro	GND	GND
Verde	Eje X	#D10
Amarillo	Eje Y	#D9
Blanco	Eje Z	#D6

Los LEDs son de alta luminosidad. Los dispositivos de señalización tienen forma de flecha, apuntando a derecha y a izquierda respectivamente.

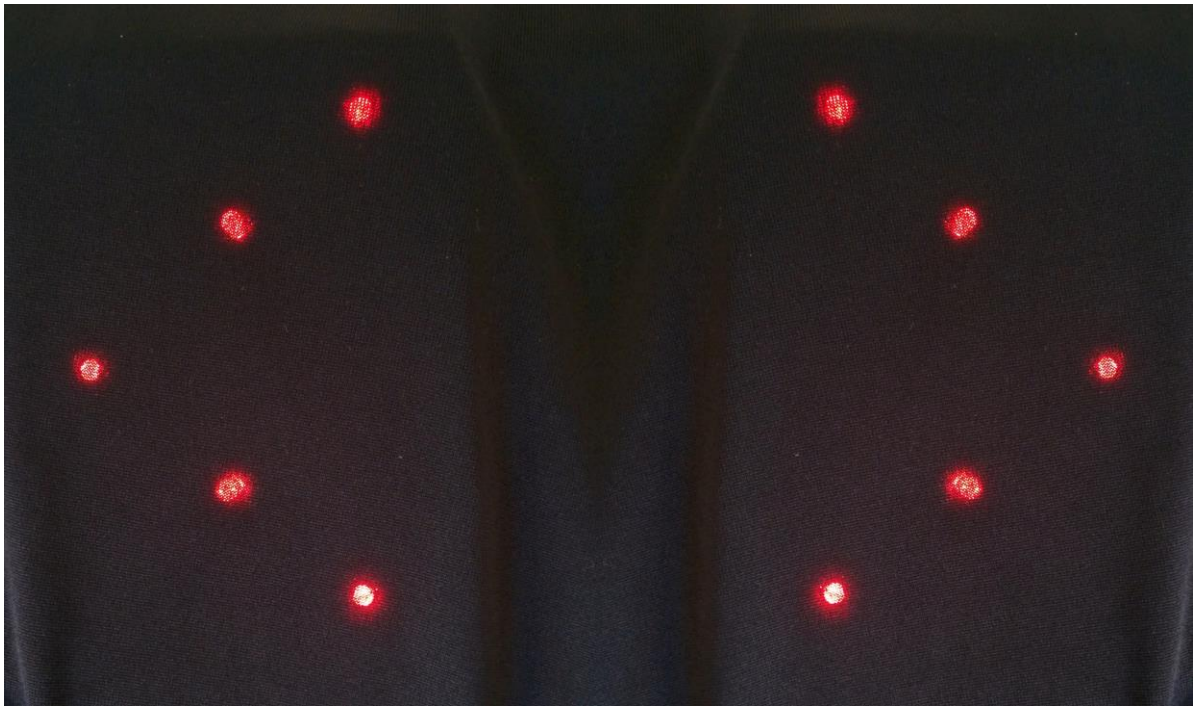


Figura 4-18. Dispositivos de señalización

Mediante hilo negro de algodón, se han fijado los dispositivos señaladores a la camiseta. Utilizando el hilo conductor y siguiendo el patrón indicado en la Figura 4.11, se ha hecho el conexionado con FLORA, así como las conexiones de FLORA con el acelerómetro.

El procedimiento al detalle se encuentra explicado en el Apartado 6, destinado a la Fase de pruebas, donde se describe paso a paso el montaje del prototipo aquí mostrado.

Figura 4-19. Prototipo finalizado. Vista general

5 ALGORITMO DE DETECCIÓN DE GESTOS

En este apartado se va a diseccionar el código desarrollado a lo largo del proyecto. El objetivo es clarificar la comprensión del mismo, identificando partes clave que permiten su funcionamiento. No se pretende mostrar todo el código existente, ya que este estará incorporado en los Anexos.

Hay dos partes claramente diferenciables en este proyecto desde un punto de vista software: la desarrollada para MATLAB® y el código C escrito para Arduino.

5.1 MATLAB®.

Se ha utilizado MATLAB® para captar los datos que proporciona el acelerómetro. Su función ha sido capturarlos vía Puerto Serie para trabajar con ellos de forma sencilla.

Debido a que MATLAB® implementa una serie de funciones que sirven para comparar señales, además de que trabajar con largos vectores no le supone ningún problema, se ha pensado que este programa era ideal para utilizarlo como contraste de cara a comprobar el correcto funcionamiento de los diferentes códigos que se han ido cargando en FLORA mediante el IDE de Arduino.

5.2 Código C.

Desde un principio, el objetivo último ha sido desarrollar un sistema completamente autónomo, sin necesidad de conectar FLORA a un tercer dispositivo donde recoger los datos.

Sin embargo, además del código final desarrollado para FLORA, donde se cumple este propósito, para llegar a él se han programado una serie de códigos intermedios con el fin de interactuar con MATLAB® mediante Puerto Serie. El fin ha sido trabajar en MATLAB® utilizando datos reales captados por el acelerómetro, de forma que se pudieran contrastar los datos procesados por FLORA con los datos procesados por MATLAB®, detectando errores en caso de haberlos y subsanándolos de la manera más eficiente. De otra forma no hubiera sido posible (o hubiera sido muy complejo) comprobar dónde fallaba el código implementado en FLORA o si los datos captados por ella no eran los esperados.

5.3 Desarrollo.

Se explica el código acorde al proceso seguido durante el Proyecto.

1. En primer lugar, se expone el código utilizado para captar los datos del acelerómetro.
2. A continuación, se implementa un algoritmo para calcular el parecido entre dos gestos capturados utilizando la correlación cruzada.

   Uno de los gestos será el patrón, es decir, el gesto de referencia que queremos identificar (vamos a emplear solamente el patrón del gesto de giro a la derecha hasta casi el final, cuando se incluirá el del giro a la izquierda).

   El otro será un gesto cualquiera capturado (llámese gesto "dato") y que se desea comprobar si responde al gesto de referencia o es otro diferente.

   Este código se implementa solamente en MATLAB®, utilizando una de sus funciones predefinidas y un algoritmo alternativo basado en la definición de correlación cruzada.
3. Tras esto, utilizando el mismo gesto y el mismo patrón del código anterior, se realizará la correlación en FLORA. Se comunicará mediante Puerto Serie con MATLAB®, que capturará los datos y mostrará la diferencia entre un método y otro.

4. Uno de los aspectos clave es la monitorización continua de los gestos realizados por el ciclista con el brazo. Para ello, es necesario añadir, al vector que guarda dichos gestos, los datos que se van recogiendo y que irán desplazando los datos ya existentes. El cuarto código es el encargado de implementarlo y de ver cómo este desplazamiento afecta al valor de la correlación cruzada, es decir, al parecido entre el gesto y el patrón.
5. Posteriormente se pasa a capturar el gesto en tiempo real, procesar la correlación cruzada en FLORA y enviar por Puerto Serie los datos a MATLAB®. El patrón estará cargado en la memoria de FLORA.
6. Como penúltimo paso, pasamos a procesar todo en la placa y a encender un LED en caso de que haya coincidencia entre el patrón y el gesto "dato".
7. Finalmente se carga también el gesto de girar a la izquierda y se amplía el código para incorporarlo al algoritmo.

De esta forma concluye el apartado destinado al desarrollo software.

5.3.1 Código 1. Captura de un gesto.

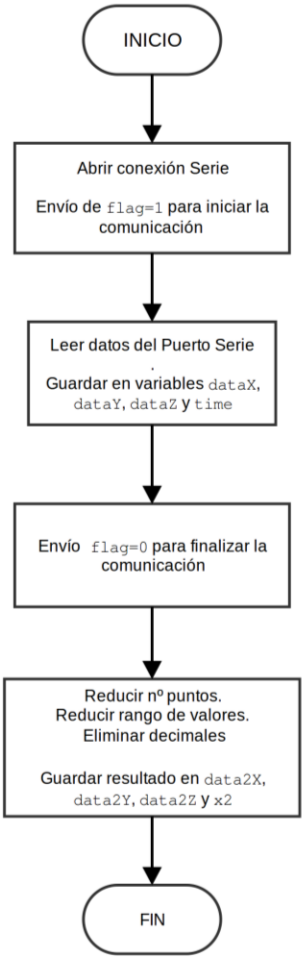
Este primer código resuelve el problema de hacer pruebas con los datos reales del acelerómetro.

Se va a implementar la comunicación entre FLORA y MATLAB® mediante el envío, por Puerto Serie, de los datos capturados por FLORA procedentes del acelerómetro. Estos datos son analógicos y corresponden a una tensión eléctrica comprendida entre 0V y 3.3V, pero serán procesados por el CAD de FLORA, obteniendo la equivalencia a esa tensión en un rango de valores que va de 0 a 1023. Estos valores se conocen como ADU (analog-to-digital units) o DN (data numbers) [23].

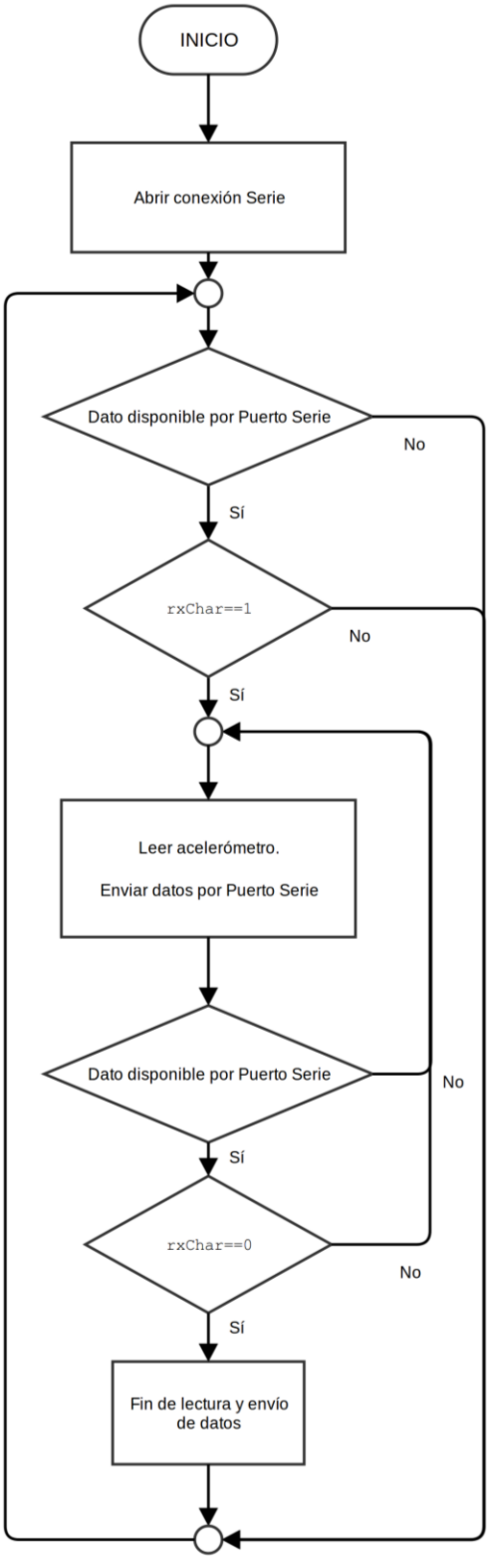
El CAD de FLORA es de 10 bits. Esto quiere decir que, con un voltaje de referencia de 3.3V, el intervalo entre 0 y 3.3V se dividirá en 1024 valores, asignando linealmente cada valor a una tensión. Por ejemplo, a la tensión media, 1.65V, se le asignará el valor 512, a 3.3V se le asignará 1023 y a 0V el 0. [24]

Hay que tener en cuenta que en una primera aproximación no importa el rango de valores pues se va a trabajar con ellos en MATLAB®, pero, como se verá posteriormente, será necesario truncar el rango y redondearlo, disminuyéndolo al comprendido entre 0 y 255, debido a las limitaciones de memoria de FLORA.

El diagrama de flujo de los códigos creados para MATLAB® y FLORA se muestra en la Figura 5-1 y dará una idea general de su funcionamiento. Es algo que se va a poner siempre antes de pasar a explicar los detalles de cada código, con el fin de mejorar su comprensión.



a)



b)

Figura 5-1. Diagrama de flujo del Código 1 para MATLAB® (a) y FLORA (b).

Se comienza la explicación del código con la apertura de conexión en MATLAB®.

script_captura_patron.m

```
%Apertura de conexion: Puerto COM4. Velocidad: 115200bps
a = serial('COM4', 'BaudRate', 115200);
fopen(a);

(...)
```

Ahora, con MATLAB® ya preparado, se procede a enviar los datos desde FLORA. Primero hay que abrir también la conexión serie, y seguidamente declarar como entradas todos aquellos los pines donde está conectado el acelerómetro:

captura_patron.ino

```
(...)

void setup() {
  // Inicializacion de comunicacion serie a 115200 bits por segundo:
  Serial.begin(115200);
  pinMode(A10, INPUT);
  pinMode(A9, INPUT);
  pinMode(A7, INPUT);
}

(...)
```

A continuación, se sincroniza la recepción de los datos en MATLAB® con el envío de los datos desde FLORA. Utilizamos una variable llamada `rxChar` en FLORA que hará las veces de *flag* o bandera. Cuando se envíe un '1' desde MATLAB®, FLORA empezará a leer datos del acelerómetro y a enviarlos. También enviará el instante (en la variable `time`) en el que realiza las capturas de los datos.

captura_patron.ino

```
(...)

void loop() {
  //Si MATLAB envia un dato, se lee y se ve si es la
  //señal asociada al comienzo de la transmisión (un '1')
  if(Serial.available())
  {
    rxChar = Serial.read();
    if(rxChar=='1')
    {
      //En el momento en el que MATLAB mande un '0', se detiene.

      while(rxChar != '0')
      {
        //Lectura del dato de un eje del acelerómetro (el X en este caso).
        int sensorXValue = analogRead(A10);
        //Envio del dato por puerto serie
        Serial.println(sensorXValue);
        int sensorYValue = analogRead(A9);
        Serial.println(sensorYValue);
        int sensorZValue = analogRead(A7);
        time=millis();
        Serial.println(sensorZValue);
        Serial.println(time);
        if(Serial.available())
```

```
        {
          rxChar=Serial.read();
        }
      }
    }
  }
  delay(16);
}
```

Finalmente, MATLAB® recibe esos datos y los va introduciendo en un vector para cada eje.

script_captura_patron.m

```
(...)

%Envio de señal de inicio de transmision a FLORA
fprintf(a,'1');

%Captura de los datos que envía. Los vectores son de total=1200 componentes.
for i=1:total,
    dataX(i,1)=str2double(fscanf(a));
    dataY(i,1)=str2double(fscanf(a));
    dataZ(i,1)=str2double(fscanf(a));
    time(i,1)=str2double(fscanf(a));
    (...)
end


%Se le envia la señal de parada a FLORA. A partir de aqui ya no transmite.
fprintf(a,'0');
```

Tras esto, se obtienen tres vectores de 1200 componentes: `dataX`, `dataY` y `dataZ` (el vector `time` no es relevante para FLORA, a pesar de que es interesante para representar los vectores en MATLAB®).

El motivo del tamaño de esos vectores no es otro que poder capturar datos del acelerómetro durante 3.5 segundos. Esto significa que se captura un dato cada, aproximadamente 3 ms o, dicho de otra manera, que se muestrea a aproximadamente 345Hz, como se explicó en el Apartado 3, dedicado al diseño del dispositivo.

Según [5], para monitorizar movimientos en extremidades hará falta, como máximo, una frecuencia de 60Hz. La idea de tomar 1200 muestras es estar muy por encima del valor de la frecuencia de Nyquist.

Son, pues, tres vectores de 1200 variables de tipo entero (2 bytes cada una), es decir, 2400 bytes por vector, 7200 bytes entres los tres, 7.2KB por gesto.

A todo esto, tenemos que tener en cuenta que, en códigos posteriores, se añadirán los vectores correspondientes a los patrones: 3 vectores (uno por eje) para el patrón de giro a la derecha y otros tres para el de giro a la izquierda.

El total de la memoria ocupada por las variables de tipo entero correspondientes a los dos patrones y al gesto a detectar aumentará hasta 21600bytes, es decir, 21.6KB.

Como se comentaba en apartados anteriores, FLORA tiene una limitación de memoria considerable, y 21.6KB de datos son demasiados, ya que tiene una SRAM de 2.5KB.

Por ello se va ha tomado la decisión de reducir el número de componentes para cada vector, pasando de 1200 componentes a 240 componentes. Esto disminuye la frecuencia de muestreo, pasando de ~345Hz a ~69Hz, superior a los 60Hz requeridos para medir el movimiento en extremidades. También se va a reducir el rango de valores a representar, pasando de representar enteros del rango 0-1023 a hacerlo entre 0 y 255.

Lo vemos en el código siguiente aplicado al gesto detectado anteriormente, compuesto por `dataX`, `dataY` y `dataZ`.

script_captura_patron.m

```
(...)

% k va a marcar las posiciones que se van a seleccionar de los vectores de
% datos correspondientes a los ejes.

k=[1:5:1200];

% Ahora se cogen 1 de cada 5 datos de los vectores originales de 1200
% muestras, por lo que se pasa de 1200 muestras a 240 muestras. Después
% se dividen los datos entre 4, para pasar de valores entre 0 y 1023 a
% valores entre 0 y 255. Finalmente se redondean para eliminar los decimales.

data2X=round(dataX(k,1)/4, 0);
data2Y=round(dataY(k,1)/4, 0);
data2Z=round(dataZ(k,1)/4, 0);
```

Tras esto, el sistema pasa de necesitar 7.2KB por gesto (21.6KB entre los tres gestos) a requerir 1.44KB (4.32KB entre los tres gestos). Todo ello se resume en la siguiente tabla y se refleja de forma más visual en el gráfico:

Tabla 5-1. Resumen de la reducción del tamaño de los vectores de datos

	ANTES		DESPUÉS	
	Componentes	Memoria	Componentes	Memoria
Eje	1200	2.4KB	240	0.48KB
Gesto (tres ejes)	3600	7.2KB	720	1.44KB
Tres gestos (2 patrones + gesto "dato")	10800	21.6KB	2160	4.32KB

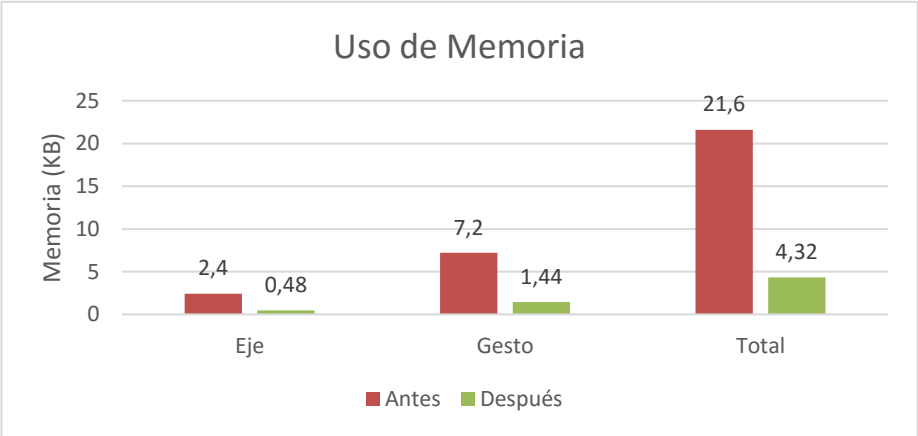


Figura 5-2. Memoria necesaria antes y después de la reducción

Los vectores siguen ocupando más que el tamaño de la SRAM, pero ahora se puede pensar en distribuirlos entre las diferentes memorias.

Como se indicará posteriormente en el Apartado 6, se pudo comprobar experimentalmente que, en el caso que nos ocupa, que la limitación en rango y tamaño no afectaba apenas a los resultados respecto al modelo con los valores obtenidos antes de la limitación, por lo que se puede hacer con menos datos sin perder información.

5.3.2 Código 2. Identificación del gesto. Cálculo del parecido.

El segundo código resuelve el problema de identificar si dos gestos se parecen (lo que sería un positivo) o si son diferentes. Con ese fin se va a utilizar la correlación cruzada.

La correlación cruzada se interpreta como una medida de la similitud lineal entre dos señales, tanto desde el punto de vista temporal como morfológico. Consiste en desplazar una señal respecto a la otra, buscando el máximo de parecido entre ambas. Es decir, va a indicar, en función del desplazamiento producido, dónde esas dos señales son más similares y su grado de similitud.

En concreto se va a utilizar la Correlación de Pearson [25]. Mide el grado de covariancia entre variables relacionadas entre sí linealmente. Este índice es independiente de la unidad de medida y el valor de las variables.

A la hora de definirla, teniendo en cuenta dos señales, $x[n]$ e $y[n]$ de la misma longitud N, el coeficiente de correlación de Pearson r_{xy} se calcula como:

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \qquad (1)$$

El valor del coeficiente de correlación de Pearson se va a mover entre -1 y 1.

- En los extremos, $r = 1$ ó $r = -1$, se encuentra un 100% de parecido entre las dos señales (caso que nunca se va a dar empíricamente), estando las señales directamente relacionadas (caso $r = 1$) o inversamente relacionadas (caso $r = -1$). En ambos casos existe una dependencia total entre ambas variables.
- En el intervalo $0 < r < 1$ se dice que el parecido entre las señales existe, aunque claramente a valores más cercanos al 1, el parecido entre ambas señales es mayor. Es decir, la correlación es positiva.
- En el intervalo $-1 < r < 0$, existe una correlación negativa.
- Si el coeficiente de correlación es 0, no existe relación lineal. Puede decirse que las dos señales no se parecen linealmente.

Un valor suficientemente grande (cercano a 1) indica un alto parecido entre la señal recibida y la referencia con la que se la compara. Es lo que se trata de encontrar mediante el código que se va a explicar a continuación, si bien, debido a que el gesto a realizar con el brazo nunca va a ser exactamente igual que el patrón, se realizarán concesiones de cara al mínimo valor de la correlación cruzada requerido para tomarlo como una identificación "positiva" del gesto buscado.

Teniendo esto en cuenta, se parte de dos señales capturadas utilizando el Código 1. Una será el patrón de girar a la derecha (mostrado en la Figura 1-2) y otra será el gesto "dato" a comparar.

Ese gesto "dato", que va a llamarse de ahora en adelante "gesto derecho", va a ser un gesto muy parecido al patrón derecho, con el objetivo de ver si este método de identificación de gestos es válido y genera un valor del coeficiente de correlación lo suficientemente alto para considerarlo correcto.

El cálculo se realizará en MATLAB® de dos maneras. La primera será mediante una función ya implementada en la biblioteca del programa con ese propósito, llamada `corr2`. La otra forma será utilizando un algoritmo alternativo basado en la definición de coeficiente de correlación de Pearson. Finalmente se compararán los resultados para ver la exactitud del método alternativo respecto al primero.

Ambos métodos se recogen en el script de MATLAB® llamado **script_cc_matlab_vs_c.m**.

El diagrama de flujo de su funcionamiento es el mostrado en la Figura 5-3.

Figura 5-3. Diagrama de flujo del Código 2

Como se puede observar, se trata de un script muy secuencial. Al iniciar carga el gesto y el patrón. A continuación, calcula de ambas formas la correlación y finalmente muestra por pantalla el resultado para poder comprobarlo.

5.3.2.1 Función `corr2`.

La correlación cruzada normalizada se calcula en MATLAB® utilizando la función `corr2` [26]. Primero se carga el patrón y el gesto "dato" y luego se hace uso de la función anteriormente indicada.

Hay que tener en cuenta que `corr2` recibe los dos vectores a comparar y devuelve el valor del coeficiente de correlación normalizada, es decir, devolverá un coeficiente cuyo valor estará comprendido entre -1 y 1.

Lo vemos en el siguiente fragmento de código.

script_cc_matlab_vs_c.m

```
(...)

load gesto.mat
load patron.mat

% Se han cargado tanto el gesto como el patrón. A continuación, hacemos uso de
% la función corr2 para calcular la correlación cruzada normalizada.

A=corr2(patronDerX,gestoDerX);
B=corr2(patronDerY,gestoDerY);
C=corr2(patronDerZ,gestoDerZ);

(...)
```

5.3.2.2 Método alternativo.

Ahora se va a buscar el valor de la correlación cruzada mediante un algoritmo alternativo [27]. Éste no es más que la definición de correlación de Pearson (1) adaptada a lenguaje MATLAB®.

Para evitar una excesiva acumulación de código en este apartado, se va a reflejar tan sólo el cálculo de la correlación en un eje, el X. La correlación de los dos ejes restantes, así como el código completo de este apartado, se podrá encontrar en los Anexos.

Importante apuntar que el nombre de las variables tiene que ver con su función dentro de la fórmula (1), así como el prefijo X para reflejar que son las variables correspondientes a la correlación del eje X. También notar que `gestoDerX` y `patronDerX` serían las correspondientes **x** e **y** en la fórmula teórica.

script_cc_matlab_vs_c.m

```
(...)

n=240;

%Variables a utilizar por el algoritmo alternativo

Xxsquare=zeros(240,1);
Xysquare=zeros(240,1);
Xxy=zeros(1,240);
Xxsum=0;
Xysum=0;
Xxysum=0;
Xxsqr_sum=0;
Xysqr_sum=0;

(...)



%Correlacion usando método alternativo
%Comienzo del bucle. En el se resuelven los sumatorios.
```

```
for i=1:n,
    Xxy(i)=gestoDerX(i)*patronDerX(i);
    Xxsquare(i)=gestoDerX(i)^2;
    Xysquare(i)=patronDerX(i)^2;
    Xxsum=Xxsum+gestoDerX(i);
    Xysum=Xysum+patronDerX(i);
    Xxysum=Xxysum+Xxy(i);
    Xxsqr_sum=Xxsqr_sum+Xxsquare(i);
    Xysqr_sum=Xysqr_sum+Xysquare(i);
end

%Finalmente se calculan numerador y denominador, y se calcula el coeficiente.

Xnum=1.0*((n*Xxysum)-(Xxsum*Xysum));
Xdeno=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
coeffXMatlab=Xnum/sqrt(Xdeno)

(...)
```

5.3.2.3 Comparación de resultados.

En la siguiente tabla resumen se recogen los valores devueltos en MATLAB® tras ejecutar el script **script_cc_matlab_vs_c.m** y también el script **script_cc_izq_matlab_vs_c.m**, análogamente desarrollado para un patrón izquierdo (mostrado en la Figura 1-1) y un gesto "dato" (diferente al gesto derecho) muy similar al patrón izquierdo (por lo que se llamará "gesto izquierdo"), ambos capturados usando el Código 1, igual que ocurría con el caso anterior.

Tabla 5-2. Comparativa de resultados de Correlación Cruzada

	Gesto Derecho		Gesto Izquierdo	
Eje	corr2	Método Alternativo	corr2	Método Alternativo
X	0.8753	0.8753	0.8486	0.8486
Y	0.8600	0.8600	0.7698	0.7698
Z	0.9364	0.9364	0.8164	0.8164

Como puede observarse, no existen diferencias cuantitativas entre ambos métodos.

5.3.3 Código 3. Correlación cruzada en FLORA.

En este apartado se va a analizar el código correspondiente a la implementación de la correlación cruzada en FLORA mediante el método alternativo, utilizando los mismos vectores patrón-gesto.

Para probar que el código programado en C y cargado en FLORA en este apartado funciona igual que el probado en MATLAB® anteriormente, va a establecerse una comunicación por Puerto Serie igual que en el Código 1 (se omitirá la apertura de la conexión en la explicación de este código). Por lo tanto, una vez más, va a haber dos desarrollos: el que va a ejecutarse en MATLAB® y el que se carga en FLORA.

El diagrama de flujo de ambos desarrollos se muestra en la Figura 5-4.

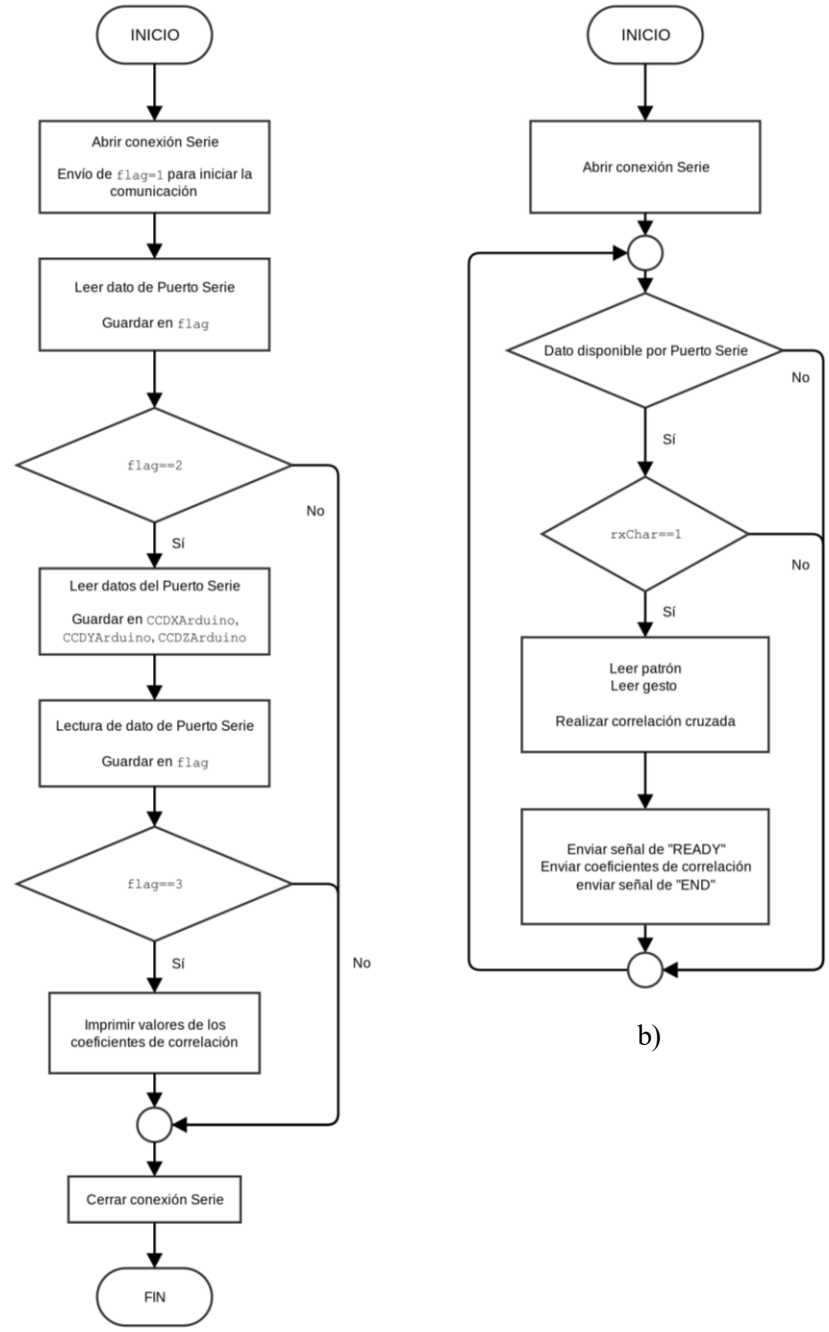


Figura 5-4. Diagrama de flujo del Código 3 para MATLAB® (a) y FLORA (b)

Como se explicó con anterioridad, debido al tamaño de la SRAM, no es posible crear ambos vectores en tiempo de ejecución, sino que se van a colocar en la memoria de programa (Flash). Para ello se utiliza la directiva **PROGMEM**, incluida en la librería **pgmspace.h** [28].

Su prototipo es el siguiente:

```
const dataType variableName[] PROGMEM = {data0, data1, data3...};
```

A la hora de leer los datos de las variables situadas en memoria de programa, hay que hacerlo en dos pasos,

leyendo de la memoria de programa para llevar la información a la SRAM y posteriormente, ya sí, utilizando esa variable en el código con normalidad.

Al igual que en el código anterior, la explicación va a centrarse en el eje X, pudiéndose consultar el código al completo en los Anexos.

cc_arduino.ino

```
#include <avr/pgmspace.h>
#include <stdint.h>
#include <math.h>

//Creación de variables 'GiroDer' y 'gesto' con los datos capturados.
const uint16_t GiroDerX[] PROGMEM = {85, 86, 85, 86, 85, 86, 86, (...) , 79, 82, 84, 86};
(...)
const uint16_t gestoX[] PROGMEM = {86, 86, 87, 86, 88, 88, 88, (...) , 86, 85, 84, 83};
(...)

//Declaración de constantes.
int i = 0;
int n = 240;
double varX=0;
double patronX=0;
double Xxy=0;
double Xxsquare=0;
double Xysquare=0;
double Xxsum=0;
double Xysum=0;
double Xxysum=0;
double Xxsqr_sum=0;
double Xysqr_sum=0;
double numX=0;
double denoX=0;
double coeffX=0;

(...)


//Declaración de las variables a utilizar en la comunicación Serie.
int READY = 2;
int END = 3;
char rxChar;


//-------------- INICIO --------------
//Comprobación de que MATLAB está preparado.
if(Serial.available())
   {
   rxChar = Serial.read();
   if(rxChar=='1')
   {


   //Inicio de la Correlación.
   for (i=0; i<n; i++)
   {
       //Lectura del patrón desde memoria de programa
       patronX=pgm_read_word_near(GiroDerX + i);
       (...)

       //Lectura del gesto desde memoria de programa
       varX=pgm_read_word_near(gestoX + i);
       (...)

       //Iteración 'i' de la correlación cruzada del eje X
       Xxy= varX * patronX;
       Xxsquare = varX * varX;
```

```
        Xysquare = patronX * patronX;
        Xxsum = Xxsum + varX;
        Xysum = Xysum + patronX;
        Xxysum = Xxysum + Xxy;
        Xxsqr_sum = Xxsqr_sum + Xxsquare;
        Xysqr_sum = Xysqr_sum + Xysquare;
        (...)

  }
  //Coeficiente Correlación Cruzada X: GESTO GIRO DERECHA
  numX=1.0*((n*Xxysum)-(Xxsum*Xysum));
  denoX=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
  coeffX=numX/sqrt(denoX);
  (...)

  //Envío de la información a MATLAB
  Serial.println(READY);
  Serial.println(coeffX);
  (...)
  Serial.println(END);

//-------------- FIN --------------
```

Primero se declaran las variables y se colocan los vectores en la memoria Flash. A continuación, se comprueba si MATLAB® está preparado para recibir los datos. Cuando lo está, se van leyendo los datos de los vectores con la función `pgm_read_word_near()`, asignándolos a variables y realizando las operaciones correspondientes.

Finalmente se calcula el dato del coeficiente de correlación y se envía a través de Puerto Serie.

Mientras tanto, en MATLAB® se está ejecutando el siguiente script:

script_cc_arduino.m

```
(...)

total=240;
flag=0;
fprintf(a,'1');
while (flag~=2)
    flag=str2double(fscanf(a));
end
CCDXArduino=str2double(fscanf(a));

(...)
```

Envía un '`1`' a FLORA y se queda a la espera de recibir un '`2`', que en FLORA hemos llamado `READY`. Cuando lo recibe, vuelve a leer un dato (el valor de la variable `coeffX` de FLORA) y almacena este valor en `CCDXArduino`.

Los datos que envía FLORA a MATLAB® se recogen en la siguiente tabla. Al igual que en el caso del Código 2, se ha hecho un código y un script análogos a los anteriores para el gesto y el patrón izquierdos, llamados **cc_arduino_izq.ino** y **script_cc_arduino_izq.m**.

Por ello, la columna izquierda se refiere al coeficiente de correlación devuelto por FLORA tras hacer la correlación del gesto derecho y patrón derecho, mientras que los valores de la columna derecha es el resultado de la correlación entre el gesto izquierdo y el patrón izquierdo.

Tabla 5-3. Valores de los coeficientes de correlación en FLORA

Eje	Gesto Giro Derecha	Gesto Giro Izquierda
X	0.88	0.85
Y	0.86	0.77
Z	0.94	0.82

Las variaciones que se observan respecto a los valores de la Tabla 5-2 son debidas a redondeos que se arrastran desde el momento en el que se disminuyó el rango de los valores en los vectores. Al realizar esta disminución o escalado, es posible que surjan algunos decimales, ya que no todos los valores entre 0 y 1023 son divisibles entre 4. Como trabajar con decimales es muy poco eficiente computacionalmente, se tomó la decisión de redondear utilizando la función `round()` definida para MATLAB®, eliminando esos decimales e, indirectamente, alterando la precisión de los valores de los vectores. Sin embargo, la pérdida de decimales no afecta significativamente desde un punto de vista de funcionamiento del sistema o a la hora de detectar el grado de similitud de dos señales.

5.3.4 Código 4. Probando el desplazamiento.

A la hora de automatizar la lectura de los datos del acelerómetro y buscar el parecido entre los datos recibidos y los patrones, hay que tener en cuenta que se trata de una aplicación en tiempo real. Sin embargo, los vectores de datos (tanto patrones como el gesto) son limitados a 240 muestras.

Si la decisión fuera tomar 240 muestras del gesto que se está haciendo en ese momento, colocarlas ordenadamente en el vector reservado para los datos del gesto, calcular el parecido, descartar las muestras y tomar 240 nuevos datos, se trataría de un proceso lento, a la par que ineficiente por desechar muestras intermedias que pueden marcar la diferencia entre un positivo o la falta total de parecido entre gesto y patrón.

Un método para ir captando los datos del gesto en tiempo real sin tener que sacrificar demasiadas muestras, es el desplazamiento de datos dentro del vector.

Consiste en tomar 240 muestras, colocándolas ordenadamente en el vector reservado para los datos del gesto, calcular el parecido entre ese vector y el patrón, desplazar los datos del vector del gesto una serie de posiciones hacia la izquierda, tomar del acelerómetro tantas nuevas muestras como posiciones desplazadas, colocarlas en el vector ordenadamente y, finalmente, volver a calcular el parecido.

Se trata de un proceso cíclico que se repite indefinidamente en el tiempo y que permite analizar de forma continua (en lugar de por bloques) el gesto realizado por el usuario de la aplicación.

El objetivo de este apartado es explicar, precisamente, el código que permite realizar estos desplazamientos.

Este código utiliza, una vez más, el mismo gesto y el patrón derecho usados en los códigos 2 y 3. De nuevo vuelve a calcular la correlación del eje X en función de esos vectores, aunque esta vez la realizará tras cada desplazamiento enviándola vía Puerto Serie, de forma que se vea cómo afecta el desplazamiento de valores al valor del coeficiente de correlación.

Para recibir estos datos, al igual que en los apartados anteriores, se va a crear, además del archivo **.ino** correspondiente para programar FLORA, un script de MATLAB®.

El resultado de estos desplazamientos se recogerá en una tabla resumen final.

Para comenzar, se muestra el diagrama de flujo correspondiente a este código en la Figura 5-5.

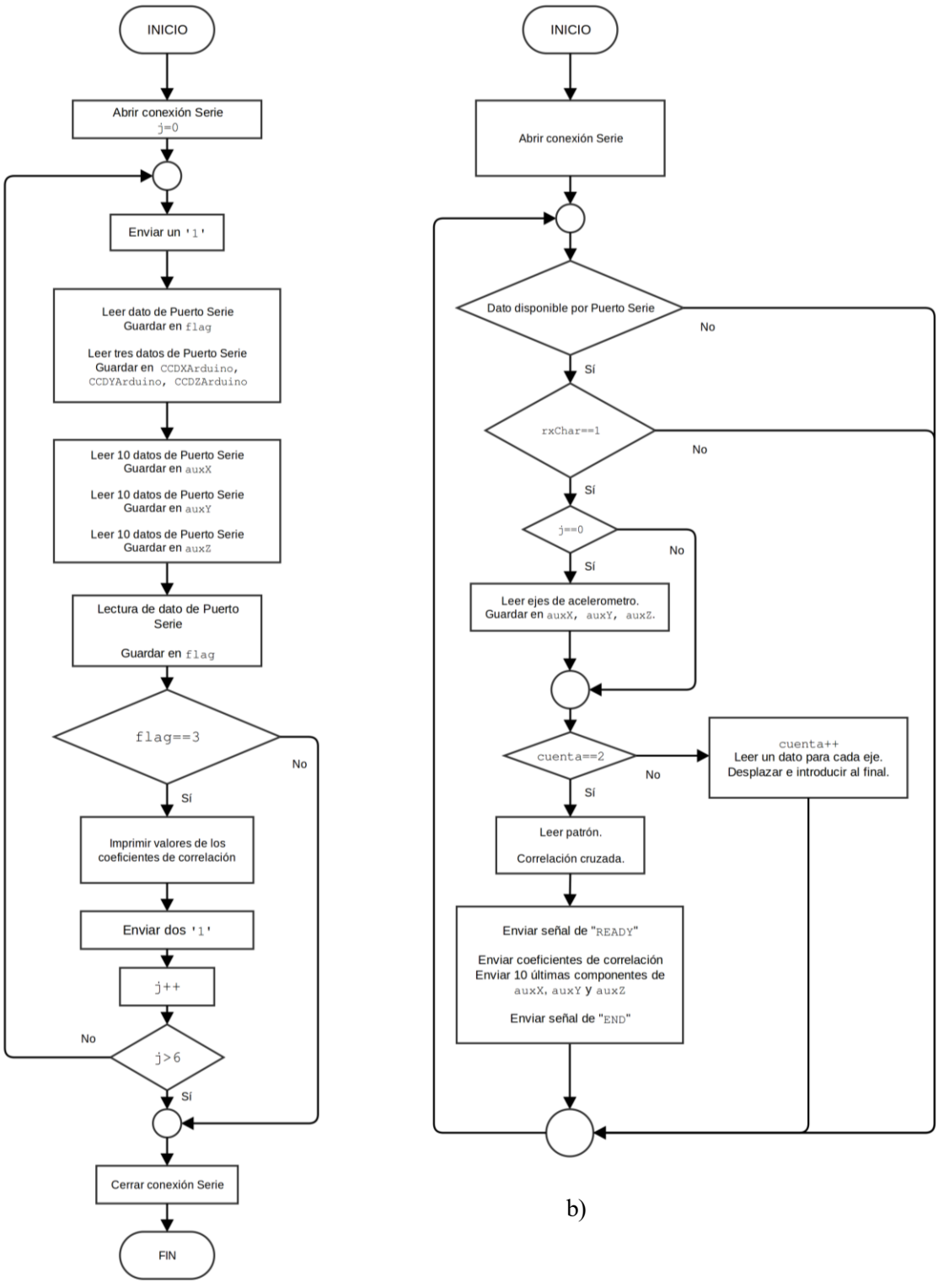


Figura 5-5. Diagrama de flujo del Código 4 para MATLAB® (a) y FLORA (b)

El código desarrollado para FLORA es:

despl_cc_arduino.ino

```
(...)

if(Serial.available())
{
   rxChar = Serial.read();
   if(rxChar=='1')
   {
     if (j==0)
     {
        //Al ser la primera vez que se ejecuta, los vectores de los ejes correspondientes
        // al gesto están vacíos. Se rellenan.
         for(i=0;i<n;i++)
         {
            auxX[i]=pgm_read_word_near(gestoX + i);
         }
         j=1;
     }
     //Comparación entre el vector de datos y el patrón cada 2 desplazamientos.
     if (cuenta==2)
     {
       (…)
       for (i=0; i<n; i++)
        {
           //Lectura del patrón
           patronX=pgm_read_word_near(GiroDerX + i);
           //Cálculo de los componentes de la fórmula de la correlación entre auxX[i] y
           //patronX.
           (...)
        }
       //Cálculo del coeficiente de correlación.
       numX=1.0*((n*Xxysum)-(Xxsum*Xysum));
       denoX=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
       coeffX=numX/sqrt(denoX);

       //Enviamos el resultado de la correlación y las 10 últimas componentes del vector.
       Serial.println(READY);
       Serial.println(coeffX);
       for(i=0;i<10;i++)
       {
          Serial.println(auxX[n-(10-i)]);
       }
       Serial.println(END);
       cuenta = 0;

     }else
     {
        //Si 'cuenta' es diferente a 2, aumentamos 'cuenta' y desplazamos.
        cuenta++;
        for(i=0;i<n;i++)
        {
           If(i==(n-1))
           {
              //Si estamos en la última posición del vector, la ponemos a 0.
              auxX[i]=0;
           }else
           {
              //Si no, en el resto de casos, hacemos que la posición i sea la i+1.
              auxX[i]=auxX[i+1];
           }
        }
     }

(...)
```

A continuación, desde MATLAB® se ejecuta el siguiente script:

script_despl_cc_arduino.m

```
(...)

total=240;
flag=0;
k=0;
%Se va a ejecutar 6 veces el bucle.
for j=1:6,
  %Se envía un 1 para que comience a enviar datos FLORA.
  fprintf(a,'1');
  %Se recibe flag que manda FLORA al iniciar el envío y los datos de la correlación.
  flag=str2double(fscanf(a));
  CCDXArduino=str2double(fscanf(a));
  (...)

  %Se reciben los datos de las 10 últimas posiciones de los vectores de los gestos.
  for r=1:10,
      auxX(1,r)=str2double(fscanf(a));
  end

  (...)

  %Se lee el flag de "final de envío" y si es el esperado, se imprime tanto la
  %correlación como los 10 últimos datos.
  flag=str2double(fscanf(a));

  if(flag==3)
    k=k+1;
    disp(' ')
    disp(['Fin de la toma de datos ', num2str(k), ':']);
    disp(' ')
    disp('    Corr. Cruzada - Giro Derecha X:')
    disp(['     ', num2str(CCDXArduino)]);
    (...)
    disp(['   Gesto X (10 últimos datos):'])
    disp(['    [', num2str(auxX),']']);
    (...)
  end

  %Aquí se envían dos '1' para incrementar la variable 'cuenta' de FLORA y desplazar
  %el vector, de forma que la próxima vez que se envíe un '1' a la placa, vuelva a
  %mandar los datos.
  for i=0:2,
      fprintf(a,'1');
  end
end

(...)
```

Como se observa, una vez más se utiliza la sincronización mediante flags entre el MATLAB® y FLORA para asegurar de que los datos recibidos son correctos y corresponden exactamente a cada una de las variables que se desean recibir.

Finalmente, se resumen en una tabla los resultados arrojados por FLORA. Además del número de la toma de datos y de los valores del coeficiente de correlación correspondiente a cada eje, en la Tablas 5-5 y 5-6 se incluyen las 10 últimas posiciones de los vectores X e Y de uno de los gestos, con el fin de que se vea cómo han ido variando con cada iteración.

Tabla 5-4. Resultados de la Correlación Cruzada en FLORA con desplazamiento

Nº de la toma de datos	Correlación Cruzada		
	X	Y	Z
1	0.88	0.86	0.94
2	0.81	0.54	0.82
3	0.75	0.41	0.75
4	0.71	0.30	0.70
5	0.67	0.22	0.66
6	0.64	0.16	0.63

Tabla 5-5. Últimas posiciones del vector del eje X del gesto en función del desplazamiento

Nº de la toma de datos	Eje X
1	[82 83 86 87 86 87 86 85 84 83]
2	[86 87 86 87 86 85 84 83 0 0]
3	[86 87 86 85 84 83 0 0 0 0]
4	[86 85 84 83 0 0 0 0 0 0]
5	[84 83 0 0 0 0 0 0 0 0]
6	[0 0 0 0 0 0 0 0 0 0]

Tabla 5-6. Últimas posiciones del vector del eje Y del gesto en función del desplazamiento

Nº de la toma de datos	Eje Y
1	[150 148 145 143 145 147 146 144 143 142]
2	[145 143 145 147 146 144 143 142 0 0]
3	[145 147 146 144 143 142 0 0 0 0]
4	[146 144 143 142 0 0 0 0 0 0]
5	[143 142 0 0 0 0 0 0 0 0]
6	[0 0 0 0 0 0 0 0 0 0]

Se puede observar que, a medida que se introducen ceros por la derecha en el vector correspondiente al eje X del gesto, el parecido respecto al eje X del patrón va disminuyendo progresivamente.

Algo llamativo en la Tabla 5-4 es la rápida disminución del valor de la correlación cruzada entre el eje Y del gesto y el eje Y del patrón a medida que se produce el desplazamiento. Esto se debe a que, en el gesto, los valores del eje Y, en torno a 143 en las últimas 10 posiciones, son mayores que los del eje X, en torno a 84. Al introducir ceros a la derecha, la caída brusca de los valores hace que baje más rápido el valor de la correlación en el eje Y que cuando se introducen esos mismos ceros en el eje X (ver Tabla 5-6).

5.3.5 Código 5. Captura de gesto, correlación cruzada en FLORA y envío de datos a MATLAB®.

Una vez se ha probado que todo lo anterior funciona correctamente, es el momento de reagrupar todas las partes y desarrollar un programa algo más parecido a lo que se pretende lograr: la autonomía de FLORA. Aún así, todavía se va a realizar el envío de los datos a MATLAB® para comprobar que lo que se está procesando en FLORA es coherente y se corresponde con lo esperado, facilitando la depuración del código en caso contrario.

Al igual que en apartados anteriores, se va a comenzar con el código de FLORA y después se presentará el código de MATLAB®. En esta ocasión no se van a recoger los datos obtenidos en una tabla debido a que el gesto a comparar con el patrón correspondiente depende, en todo momento, del movimiento que se esté realizando con el brazo.

Además, si se enviaran datos tales como el gesto realizado para poder representarlo, se estaría modificando el rendimiento del dispositivo, ya que el tiempo necesario para enviar los tres vectores del gesto capturado por FLORA sería muy grande (hay que tener en cuenta que se va a estar monitorizando durante un tiempo, por lo que serían demasiados vectores a transmitir).

El patrón que se va a utilizar para la detección del gesto es, una vez más, el de giro a la derecha. Igualmente, se va a explicar código correspondiente al eje X, recogiendo el código al completo en los Anexos.

Antes de pasar a desarrollar el código, se presenta, en la Figura 5-6, el correspondiente diagrama de flujo que muestra su funcionamiento.

Lo más interesante del código a cargar en FLORA está dentro de la función periódica **loop**.

Cuando recibe la orden de MATLAB®, comienza el bucle. La primera vez, con el flag `j` forzado a'`0`', se cogen 240 datos del acelerómetro y se guardan en los vectores `aux`. Estos vectores son los que van a almacenar los gestos a comparar con el patrón. A partir de aquí, `j` va a ser '`1`' durante el resto del tiempo, por lo que no vuelve a cumplirse la condición `j==0` nunca más.

Hay que tener en cuenta también que `cuenta=20` la primera vez que se ejecuta el programa. Esta variable se dedica a contar el número de iteraciones que ha dado el loop. La condición de realizar la correlación, como se puede ver en el diagrama de flujo, es `cuenta==20`, por lo que, al estar predeterminada a `20` al iniciar la ejecución del código, con los datos captados bajo la condición `j==0` hace la primera correlación.

A partir de aquí, cuenta se pone a '`0`' y comienza el funcionamiento normal del programa. Cada 20 iteraciones del bucle, hará la correlación y la enviará a MATLAB®. Si `cuenta` es diferente a `20`, FLORA toma muestras del acelerómetro y las va metiendo al final de los vectores `aux`, como ocurría en el Código 4. Cada vez que realiza esto, `cuenta` aumenta en una unidad.

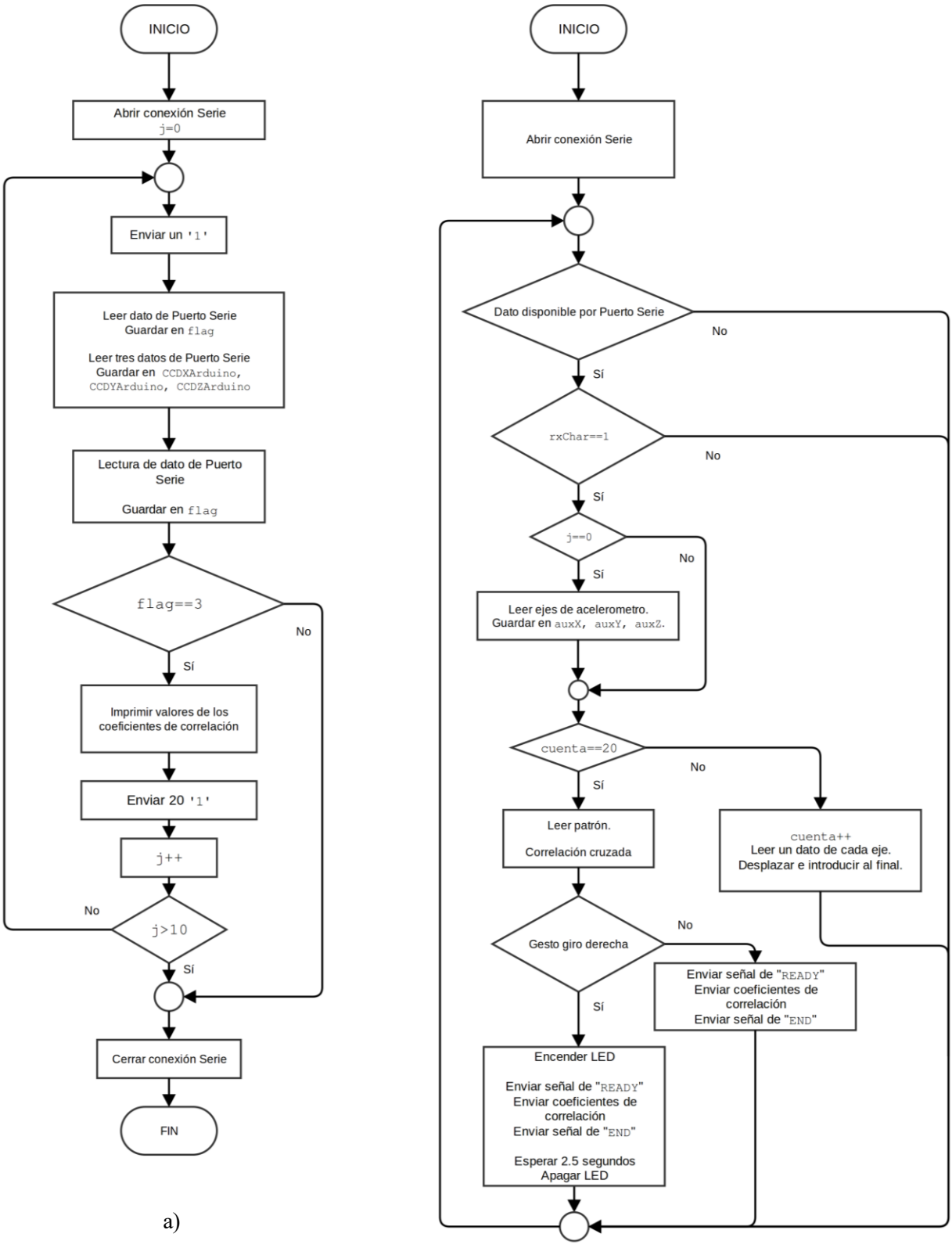


Figura 5-6. Diagrama de flujo del Código 5 para MATLAB® (a) y FLORA (b)

El código que se está ejecutando en FLORA es el siguiente. Hay que destacar la condición `if-else` para dar como válido un gesto o no.

En caso de que los coeficientes de correlación de cada eje sean superiores a 0.8 (hay que recordar que la correlación cruzada normalizada devuelve un coeficiente cuyo valor está entre -1 y 1), se va a tomar como válido el gesto.

Es un valor muy alto, pero si se está trabajando en un entorno más o menos ideal, es decir, parado y realizando el gesto con el brazo de la forma más cuidadosamente posible, sin ninguna perturbación externa, es posible replicar el patrón con mucha exactitud.

En apartados posteriores se verá que, el mínimo valor de los coeficientes de correlación necesario para dar como positiva la identificación de un gesto respecto a un patrón, se va a disminuir para que el sistema no sea tan exigente a la hora de decidir si un gesto se identifica con un patrón o no. Tomar esta decisión puede dar error a falsos positivos, por eso hay que encontrar un valor de compromiso que minimice la probabilidad de error sin restringir demasiado el gesto a realizar, dando como válidos gestos muy parecidos al patrón pero contemplando la posibilidad de una cierta variación en el movimiento del brazo.

lectura_cc_arduino.ino

```
(...)

//La primera vez, se llenan los vectores con los datos que se reciben del acelerómetro.

if (j==0)
{
  for(i=0;i<n;i++)
  {
    auxX[i]=analogRead(A10)/4;

    (...)

  }
  j=1;
}

//Comparación entre el vector de datos y el patrón cada 20 muestras.

if (cuenta==20)
{
  Xxsum=0;
  Xysum=0;
  Xxysum=0;
  Xxsqr_sum=0;
  Xysqr_sum=0;

  (...)

  for (i=0; i<n; i++)
  {
    //Lectura del patrón

    patronX=pgm_read_word_near(GiroDerX + i);
    (...)

    //Correlación cruzada eje X

    Xxy= auxX[i] * patronX;
    Xxsquare = auxX[i] * auxX[i];
    Xysquare = patronX * patronX;
    Xxsum = Xxsum + auxX[i];
    Xysum = Xysum + patronX;
    Xxysum = Xxysum + Xxy;
    Xxsqr_sum = Xxsqr_sum + Xxsquare;
    Xysqr_sum = Xysqr_sum + Xysquare;
```

```
   (...)

  }


  //Coeficiente Correlación Cruzada X: GESTO GIRO DERECHA

  numX=1.0*((n*Xxysum)-(Xxsum*Xysum));
  denoX=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
  coeffX=numX/sqrt(denoX);

  (...)


  //Ahora se ve si el resultado de la correlación se puede dar como un positivo o no.
  //Aquí se añaden también las correlaciones de los ejes Y y Z, omitidas en el resto del
  //código. El objetivo es que quede claro cómo se detecta el gesto.

  if (coeffX>0.8 && coeffY>0.8 && coeffZ>0.8)
  {
    digitalWrite(ledpin, HIGH);
    Serial.println(READY);
    Serial.println(coeffX);
    (...)
    Serial.println(END);

    delay(2500);
    digitalWrite(ledpin,LOW);
  }else
  {
    digitalWrite(ledpin, LOW);
    Serial.println(READY);
    Serial.println(coeffX);
    (...)
    Serial.println(END);
  }

  cuenta = 0;

}else
{
  //Si cuenta!=20, se lee un nuevo dato de cada eje y se mete al final del vector
  //correspondiente. Se aumenta 'cuenta'.

  cuenta++;
  for(i=0;i<n;i++)
  {
    if(i==(n-1))
    {
      auxX[i]=analogRead(A10)/4;
      (...)
    }else
    {
      auxX[i]=auxX[i+1];
      (...)
    }
  }
}
```

En el funcionamiento definitivo del sistema, no existe la necesidad de que un programa le transmita órdenes a FLORA. Va a ser un dispositivo completamente independiente.

En este caso, en el código arriba detallado, se va a realizar el bucle cada vez que se recibe un '1' enviado desde MATLAB® por Puerto Serie. El objetivo es tener más control sobre el sistema en este punto tan delicado del desarrollo, cuando FLORA empieza a procesar directamente la información que le viene desde el acelerómetro.

Se ha tomado la decisión de hacerlo así de cara a mejorar la depuración del código.

Para ilustrar este punto, se va a explicar el código MATLAB®.

lectura_cc_arduino.m

```
(...)

%Se van a realizar 10 capturas de la correlación cruzada, por eso j va de 1 a 10.
for j=1:10,
  fprintf(a,'1');
  flag=str2double(fscanf(a));

  CCDXArduino=str2double(fscanf(a));
  CCDYArduino=str2double(fscanf(a));
  CCDZArduino=str2double(fscanf(a));

  flag=str2double(fscanf(a));

  %Si este último flag==3, significa que FLORA ha enviado la señal de 'END' y ha
  %terminado el cálculo de la CC en la iteración 'k', por lo que imprimimos su valor.

  if(flag==3)
    k=k+1;
    disp(' ')
    disp(['Fin de la toma de datos ', num2str(k), ':']);
    disp(' ')
    disp('    Corr. Cruzada - Giro Derecha X:')
    disp(['     ', num2str(CCDXArduino)]);
    disp('    Corr. Cruzada - Giro Derecha Y:')
    disp(['     ', num2str(CCDYArduino)]);
    disp('    Corr. Cruzada - Giro Derecha Z:')
    disp(['     ', num2str(CCDZArduino)]);
    disp(' ')
    disp(' ')
  end

  %Ahora se envía 20 veces un '1'. Por cada uno de ellos, FLORA lee un dato del
  %acelerómetro (un dato de cada eje) y lo mete al final del vector de datos.

  for i=0:20,
    fprintf(a,'1');
  end

end
```

Como se comentaba en la página anterior, el código de MATLAB® está sincronizado con FLORA. Cada veinte '1' recibidos, FLORA realiza la correlación y la envía por puerto serie, por lo que cada 20 iteraciones MATLAB® lee los datos correspondientes y los imprime por pantalla.

5.3.6 Código 6. Captura, correlación y encendido de LED.

El Código 6 es el primer código que hace completamente independiente a FLORA. A partir de aquí ya no va a ser necesario utilizar MATLAB®, ya que la propia placa leerá los datos correspondientes al gesto del brazo y comparará dicho gesto con el patrón, encendiendo un LED al detectar que el gesto realizado coincide con el que se quiere identificar.

En la siguiente página se muestra el diagrama de flujo correspondiente (Figura 5-7).

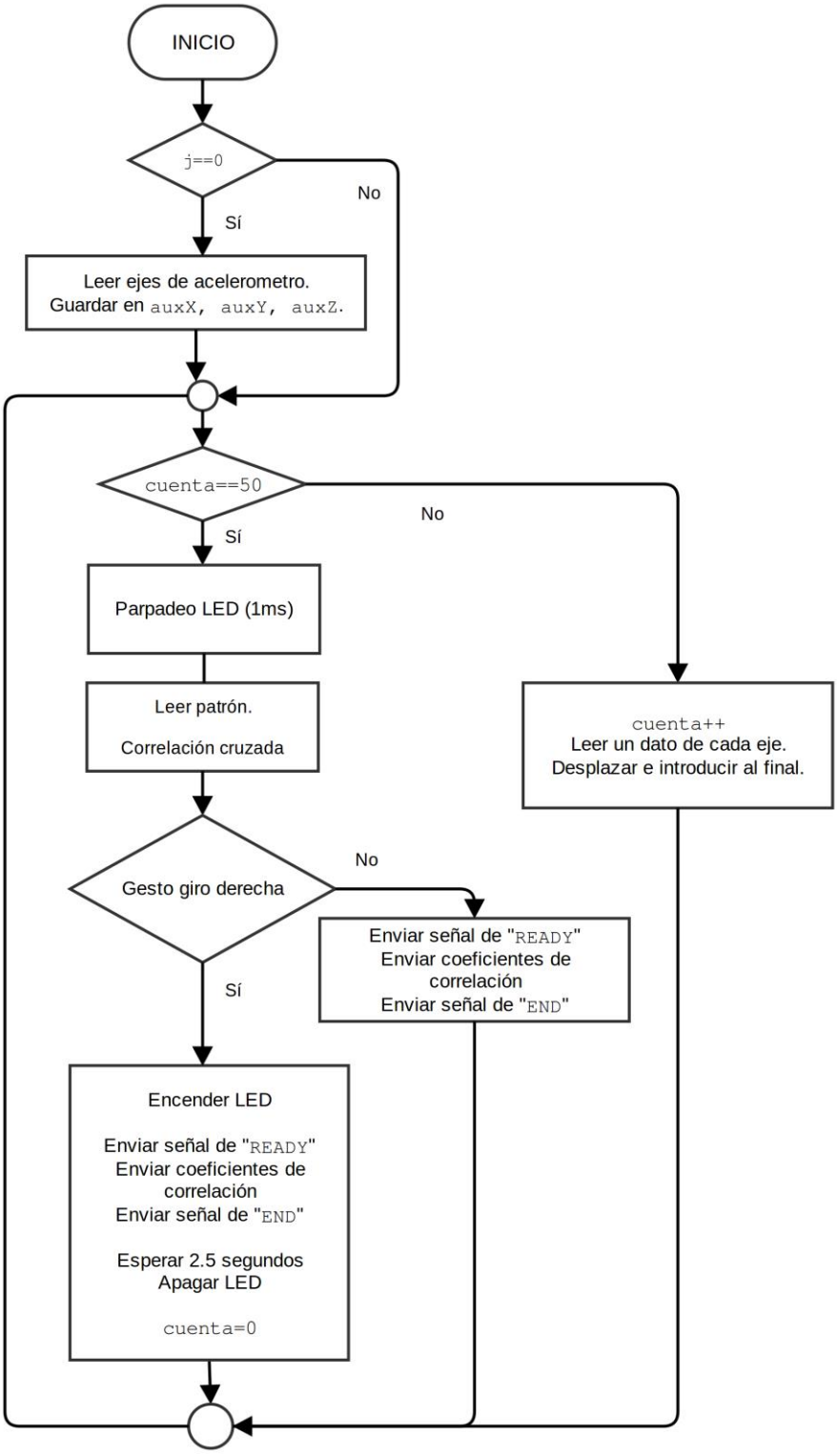


Figura 5-7. Diagrama de flujo del Código 6.

Es muy parecido estructuralmente al Código 5, por lo que no van a explicarse muchas líneas. La única diferencia radica en que se eliminan por completo las partes que hacían que el Código 5 dependiera de MATLAB®, es decir, todo lo que tenía que ver con las banderas para facilitar la sincronización y el envío de los resultados de las distintas correlaciones por Puerto Serie. Nada de eso es necesario llegados a este punto, ya que se ha comprobado que el algoritmo funciona correctamente.

Lo único a destacar es el siguiente fragmento, que es el encargado de encender el LED situado en el pin #D7

durante 3,5 segundos en caso de detectar un positivo, que en este caso será un coeficiente de correlación superior a 0.7 en cada eje.

lectura_cc_arduino_led.ino

```
(...)

//Si el parecido es notable entre el gesto realizado y el patrón, se toma como positivo.
//Se va a encender el led #D7 durante 3.5 segundos y se va a volver a leer los datos del
//acelerómetro. Luego se va a apagar.

//Aquí se baja de 0.8 a 0.7 el valor del coeficiente de correlación necesario para
//dar como positivo un gesto, como se indicó en el Apartado 5.

if (coeffX>0.7 && coeffY>0.7 && coeffZ>0.7)
{
  digitalWrite(ledpin, HIGH);
  for(i=0;i<n;i++)
  {
    auxX[i]=analogRead(A10)/4;
    auxY[i]=analogRead(A9)/4;
    auxZ[i]=analogRead(A7)/4;
  }
  delay(3500);
  digitalWrite(ledpin,LOW);
}else
{
  digitalWrite(ledpin, LOW);
}

(...)
```

El objetivo de volver a leer `n=240` datos del acelerómetro tras encontrar un positivo, es "limpiar" el vector `aux` de los datos del gesto que see acaba de comparar con el patrón satisfactoriamente.

La razón no es otra que evitar que, tras dar como positivo el gesto por ser parecido al patrón, en el siguiente cálculo del coeficiente de correlación, tras haber tomado 50 nuevos datos de cada eje y haberlos metido en el correspondiente vector `aux`, desplazando los datos ya existentes, el parecido entre el gesto y el patrón siga siendo lo suficientemente alto como para dar positivo de nuevo, a pesar de haber realizado un gesto completamente diferente. En el fondo tomamos 50 datos del nuevo gesto, pero el vector tiene 240 datos, por lo que, si no es un gesto que pueda modificar radicalmente `aux` para cada uno de los tres ejes, es posible dar un falso positivo.

Es más, incluso si, tras el positivo, se realiza un gesto exactamente igual que el patrón, es posible que, erróneamente, no lo tome como positivo. Es debido a que, al concatenar el final del gesto anteriormente detectado como válido con el nuevo gesto realizado, la señal resultante puede ser muy diferente al patrón. Es decir, a la hora de repetir el gesto, no se obtendría otro positivo. En su lugar se tendría que dejar pasar un tiempo prudencial hasta que se "limpiase" el vector para realizar un gesto como el patrón y que se tomase, correctamente, como positivo.

5.3.7 Código 7. Implementación final en FLORA.

Para tener la funcionalidad total del dispositivo, en este código se va a incorporar el patrón de giro a la izquierda, así como se van a definir los pines de salida para las señales luminosas, compuestas por LEDs, dando por finalizada la parte de diseño software.

El código 6 se ha reescrito, adquiriendo una estructura de máquina de estado, además de añadir el patrón izquierdo y la comparación del gesto capturado con el mismo. El **loop** lo preside un `switch-case` que depende del estado en el que se encuentre el dispositivo. Los estados son `NADA`, `COMPARA`, `LED_DER` y `LED_IZQ`. Las relaciones entre los distintos estados se recogen en el siguiente diagrama de estados.

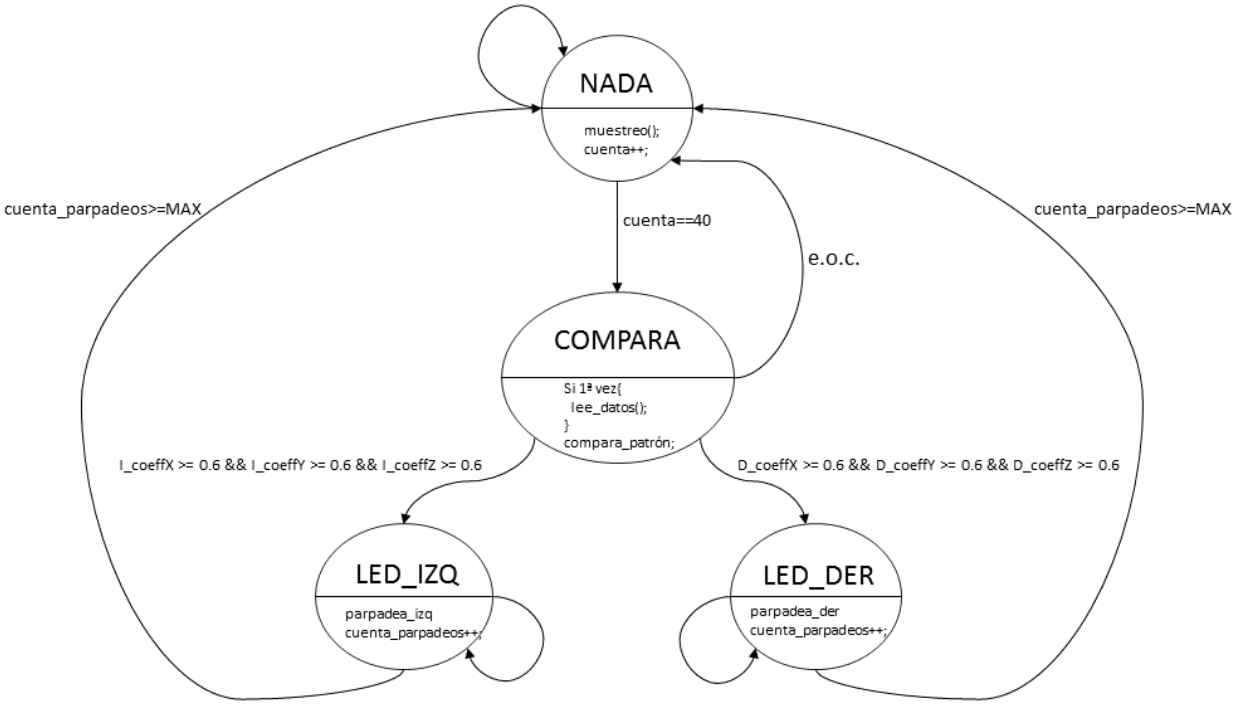


Figura 5-8. Diagrama de estados del Código 7

Para explicar el diagrama anterior, se va a seguir el flujo del programa, reflejado en la Figura 5-9.

A pesar de que `NADA` va a ser el estado en el que el microcontrolador va a pasar más tiempo, `COMPARA` va a ser el estado inicial.

`COMPARA` es el estado en el que se realiza el cálculo del coeficiente de correlación. En él se entra inicialmente, por lo que, al principio, al igual que sucedía en el Código 6, es necesario capturar 240 datos para cada eje del acelerómetro e introducirlos en el vector `aux` correspondiente, que se guarda la información del gesto que se está realizando con el brazo. Este proceso lo realiza la función `lee_datos()` sólo la primera vez. A partir de aquí, `COMPARA` se limita a realizar la correlación cruzada, comparando los patrones y el gesto mediante la función `compara_patron()`.

Si el gesto es reconocido, puede ser que sea el gesto de girar a la derecha o de girar a la izquierda. En función de la dirección, pasamos al estado `LED_IZQ` o `LED_DER`. Son estados casi gemelos, ambos realizan la misma función: encienden y apagan los LEDs de la espalda en función del sentido del giro detectado. Sólo se diferencian en el pin de salida que activan para encender sus correspondientes LEDs. `LED_IZQ` activará el pin `#D2` y `LED_DER` activará el pin `#D0`. Cuando llega al límite de parpadeos predeterminado, pasan al estado `NADA`.

Es posible también que el gesto no sea un gesto reconocible, por lo que, desde `COMPARA`, pasaría directamente a `NADA`, sin ir ni a `LED_IZQ` ni a `LED_DER`.

El estado `NADA` es el estado en el que más tiempo va a pasar FLORA. En este estado, se lee del acelerómetro usando la función `muestrea()` y se incrementa la variable `cuenta`. Cuando `cuenta==40`, se fija que el siguiente estado es `COMPARA` y el ciclo vuelve a comenzar.

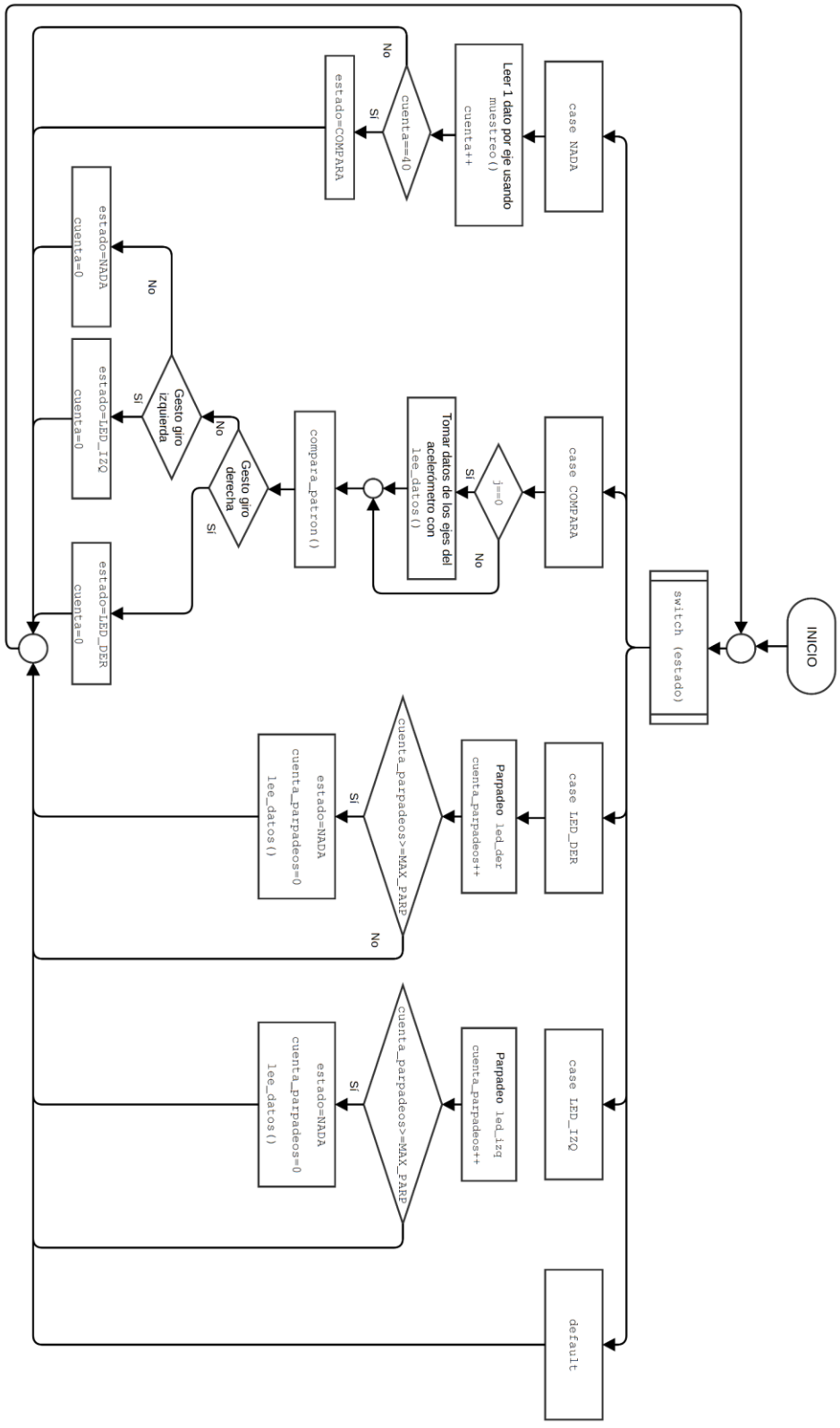


Figura 5-9. Diagrama de flujo del Código 7

Vemos el código en detalle a continuación. Se ha vuelto a disminuir el valor de los coeficientes de correlación para dar como positivo un gesto. El valor de compromiso encontrado y que mejora el reconocimiento de gestos sin caer en falsos positivos es 0.6.

tfg.ino

```
(...)
switch (estado)
{
   case NADA:
       muestreo(&cuenta, auxX, auxY, auxZ, n);
       cuenta++;
       if(cuenta==40)
       {
           estado=COMPARA;
       }
       break;

   case COMPARA:
       if (j==0)
       {
           lee_datos(auxX, auxY, auxZ,n);
           j=1;
       }

       compara_patron(auxX, (...), &DcoeffX, (...), &IcoeffX, (...), ledpin, n);
       if (DcoeffX>0.6 && DcoeffY>0.6 && DcoeffZ>0.6)
       {
           estado=LED_DER;
           cuenta=0;
       }else if (IcoeffX>0.6 && IcoeffY>0.6 && IcoeffZ>0.6)
       {
           estado=LED_IZQ;
           cuenta=0;
       }else
       {
           estado=NADA;
           cuenta=0;
       }
       break;

    case LED_DER:
        digitalWrite(led_der,HIGH);
        delay(500);
        digitalWrite(led_der,LOW);
        delay(500);
        cuenta_parpadeos++;
        if(cuenta_parpadeos >= (MAX_PARPADEOS))
        {
            estado=NADA;
            cuenta_parpadeos=0;
            lee_datos(auxX, auxY, auxZ,n);
        }
        break;

    case LED_IZQ:
        (...)

    default:
        (...)
}
```

En este apartado se ha hecho hincapié en el `switch` que preside la función **loop**, ya que va a ser donde se va a ir realizando la transición entre los diferentes estados. El resto del código faltante se ha reconvertido en las funciones `muestreo()`, `lee_datos()` y `compara_patron()`, disponibles, para su consulta, en los Anexos.

Una vez que el código anterior está completado, el objetivo de un sistema de reconocimiento de gestos independiente está cumplido.

6 INTEGRACIÓN Y PRUEBAS

Como se ha visto en el Apartado 5, muchos de los códigos incluyen una conexión por Puerto Serie entre FLORA y MATLAB®.

Los datos recibidos se han ido guardando en variables con el objetivo de trabajar con ellos de forma "bruta", procesarlos para obtener ciertas características, o ver hasta qué punto se podían simplificar para reducir la carga de trabajo del microcontrolador.

Lo primero será conectar FLORA al acelerómetro y al ordenador mediante las conexiones explicadas en el apartado 4.4.

6.1 Captura de los patrones y disminución del tamaño de los vectores.

En primer lugar, lo que se va a realizar va a ser la captura de un gesto cualquiera. Se pretende comprobar que (además de que se reciben correctamente los datos del acelerómetro en el ordenador) tras la reducción del número de componentes de los vectores de cada eje, pasando de 1200 a 240 componentes, la señal apenas se ve afectada morfológicamente.

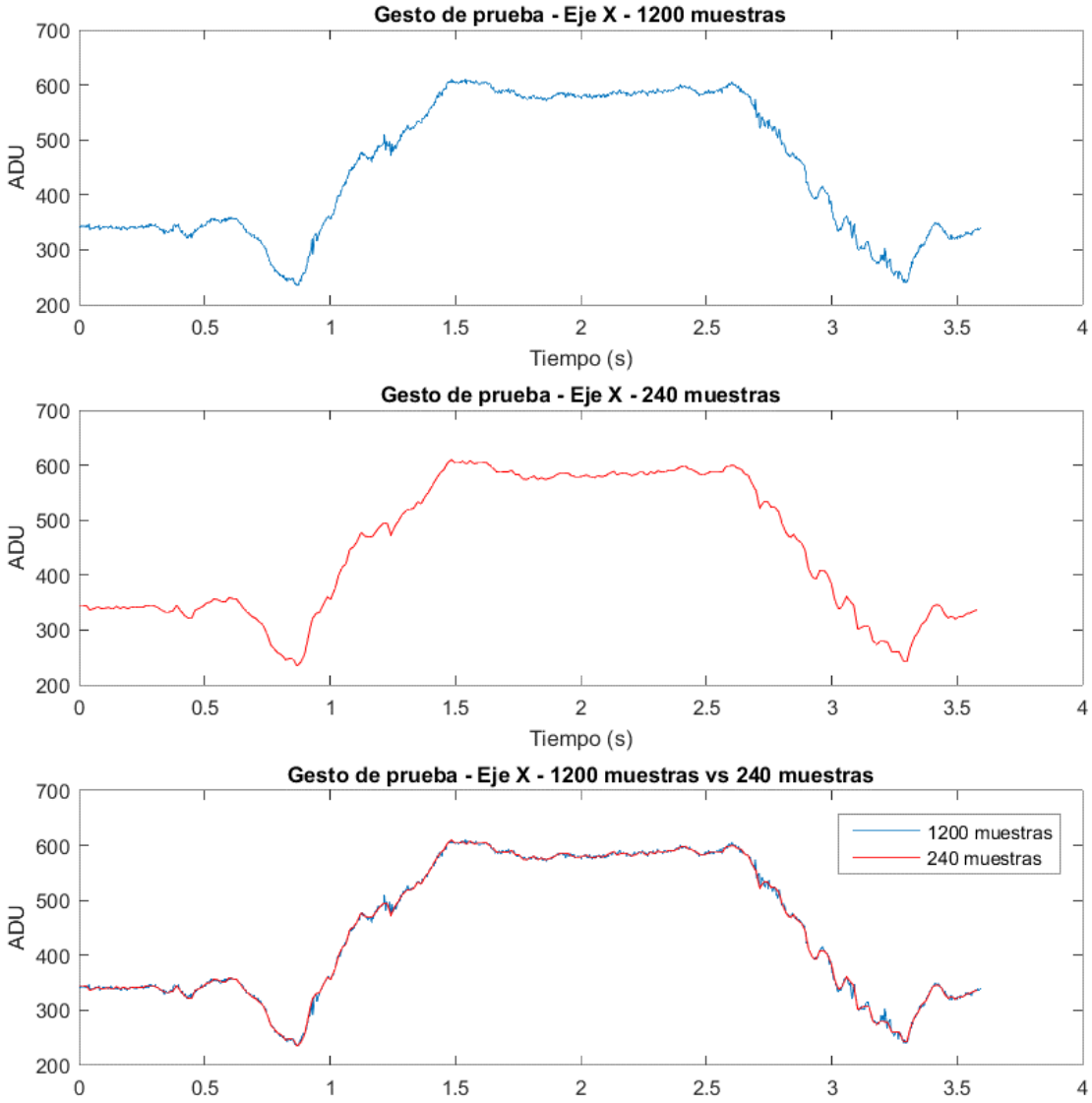


Figura 6-1. Gesto de prueba. Reducción del tamaño del vector

Si se analiza la Figura 6-1, se puede observar cómo se han eliminado pequeñas aceleraciones que daban forma de diente de sierra a la representación gráfica de la señal. Es otro aspecto positivo de la reducción del número de muestras: la reducción del ruido.

Una vez se ha comprobado ese punto, se pasa a capturar uno de los gestos que se pretenden reconocer, el de señalización del giro a la derecha (Figura 1-2). Utilizando el Código 1, se ha realizado varias veces el gesto hasta obtener uno lo suficientemente suave y definido, de cara a facilitar el contraste entre esta señal y cualquier otro movimiento.

Así, se obtiene uno de los patrones (el derecho) con los que se comparará más adelante el resto de movimientos del brazo.

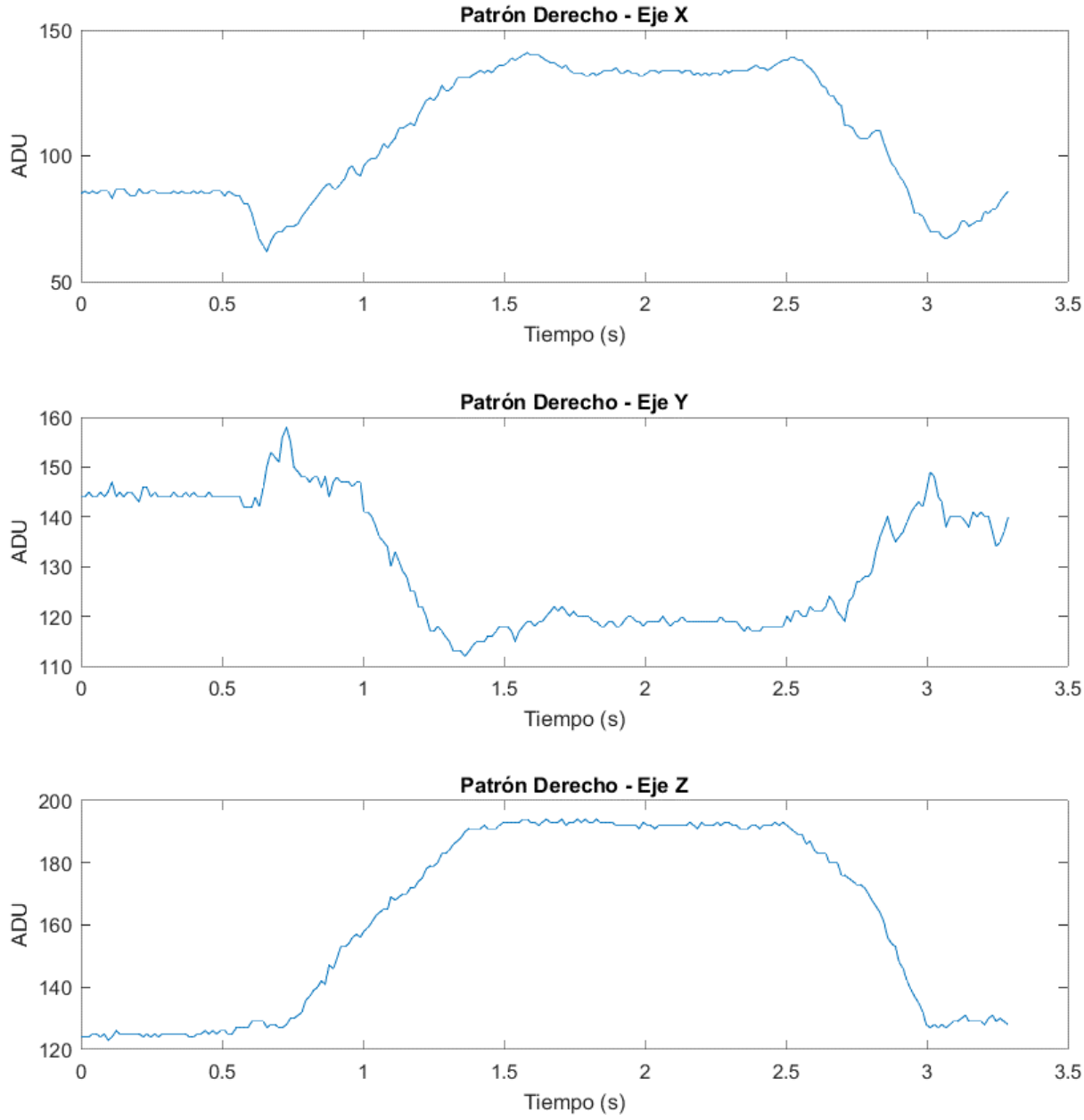


Figura 6-2. Patrón de giro a la derecha

Se puede observar en el eje vertical de las gráficas de la Figura 6-2 que se ha reducido también el rango de valores a representar, pasando de los que salen del convertidor, en el rango de 0 a 1023, a un rango más pequeño, de 0 a 255. Como se vio durante la explicación del Código 1, esta reducción también conlleva mejor uso de la memoria disponible en FLORA.

6.2 Captura de gestos y comparación con patrones.

Una vez que se tiene un patrón con el que comparar, se detecta otro gesto. Éste está realizado con la intención de que resulte lo más similar posible al patrón. Con el fin de visualizar mejor la diferencia ente el patrón derecho y este gesto, que se denominará "gesto derecho", se van a representar los dos en la misma gráfica (Figura 6-3).

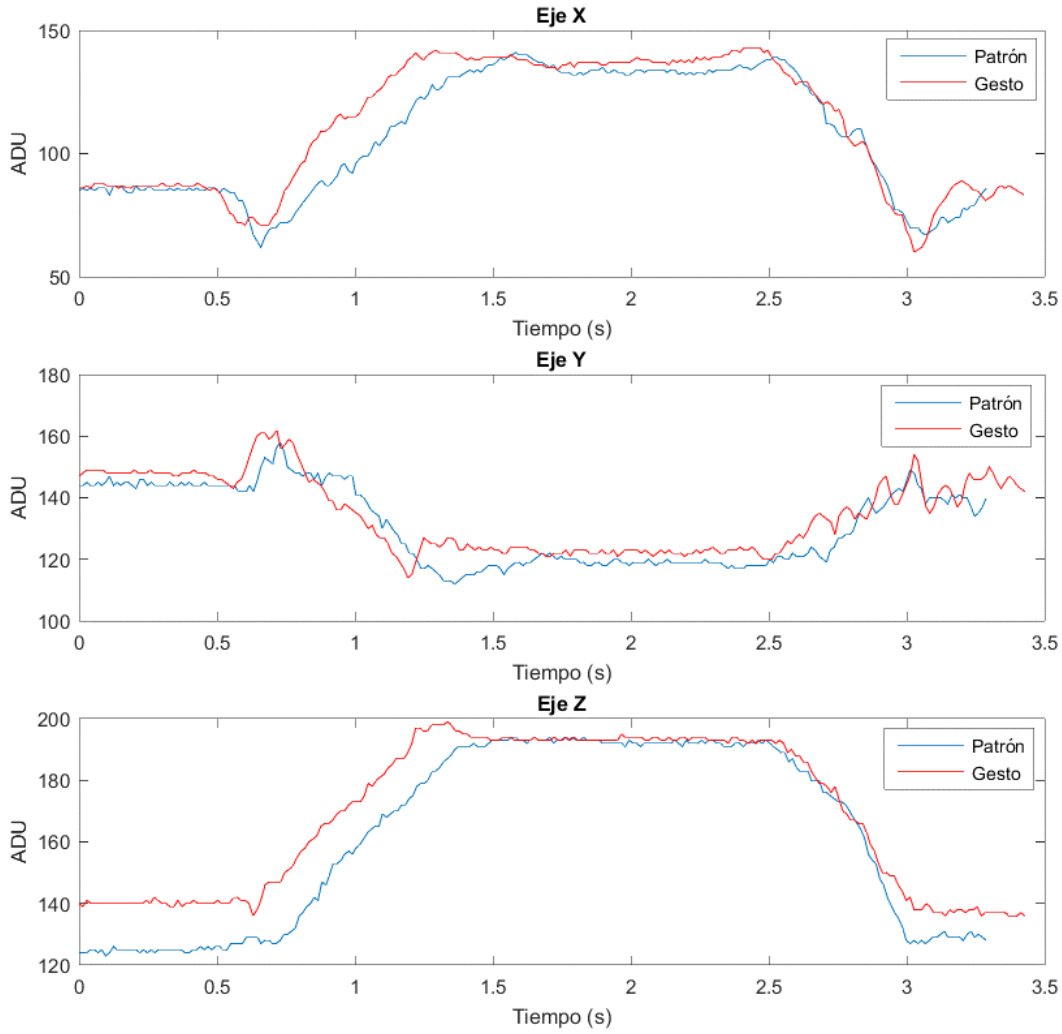


Figura 6-3. Patrón de giro a la derecha VS gesto derecho

Lo mismo se ha hecho con el gesto de girar a la izquierda. Se ha capturado un gesto acorde a la Figura 1-1, que será el patrón izquierdo, y se ha capturado a continuación otro movimiento lo más parecido posible a él. Se denominará, al igual que en el caso anterior, "gesto izquierdo".

Se muestran los resultados en la Figura 6-4.

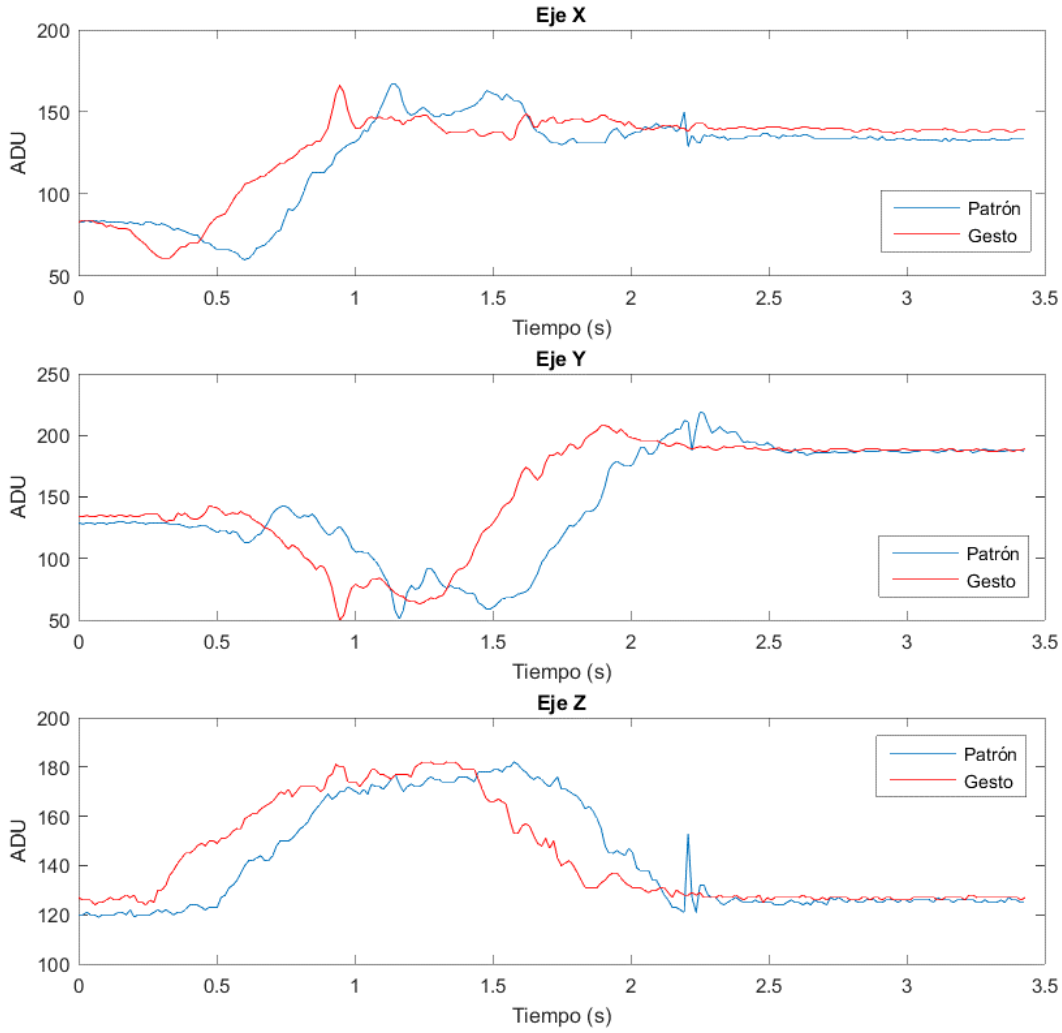


Figura 6-4. Patrón de giro a la izquierda VS gesto izquierdo

Resulta evidente que existe una correlación muy grande entre el patrón derecho y el gesto derecho, al igual que existe entre el patrón izquierdo y el gesto izquierdo.

Es ahora cuando entra en juego lo explicado en el Código 2.

El código implementado en FLORA es la definición de correlación de Pearson, y calcula el coeficiente de correlación. En MATLAB® se ha calculado el coeficiente de correlación usando la función `corr2`. Ambos resultados se han recogido en la Tabla 5-2.

Se ve que los valores de cada uno de los coeficientes de correlación, en cada uno de los casos, se acercan mucho a 1. Es lo esperado, ya que, como se ha visto anteriormente, se han realizado cada uno de los gestos para que sean lo más parecido posible a sus respectivos patrones, y cuanto más cercano a 1 es el valor del coeficiente de correlación, más parecidas son las dos señales. A valor más cercano a 1, mayor seguridad se tendrá para afirmar que el gesto corresponde a un patrón.

Algo que no se ha realizado aún, es una comparativa entre el parecido de un mismo gesto respecto a los dos patrones, el derecho y el izquierdo.

Como se comenta, se van a comparar los gestos tanto con el patrón de giro a la derecha como con el patrón de giro a la izquierda, de forma que se visualice cómo las características morfológicas y temporales de las señales correspondientes a los ejes X, Y y Z del gesto, se traducen en una variación del valor del coeficiente de correlación.

El primer paso va a ser la representación gráfica de los gestos y los patrones en una misma figura.

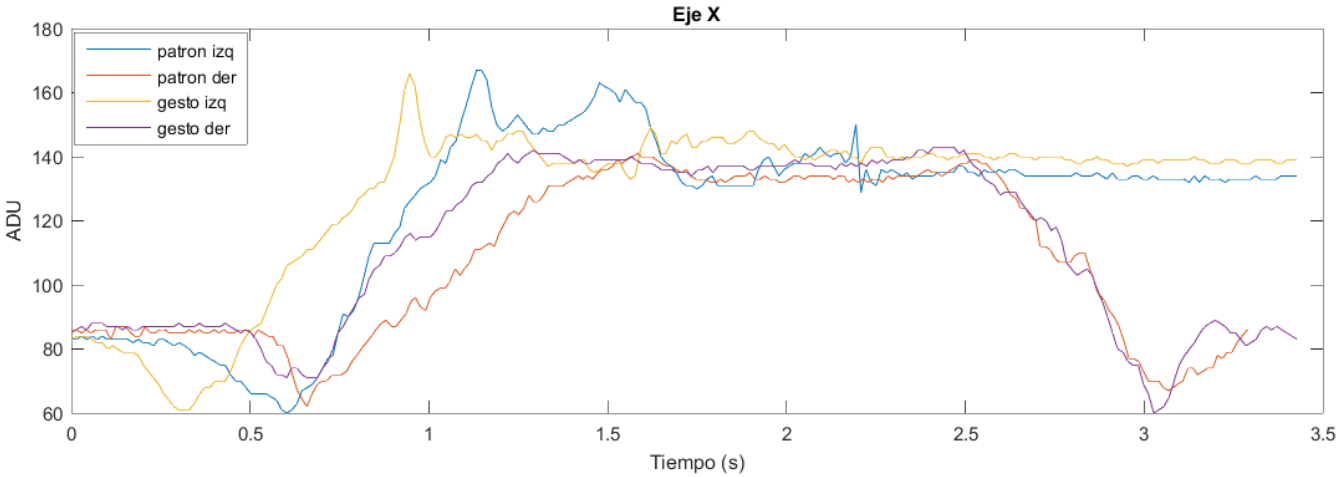


Figura 6-5. Patrones y gestos - Eje X

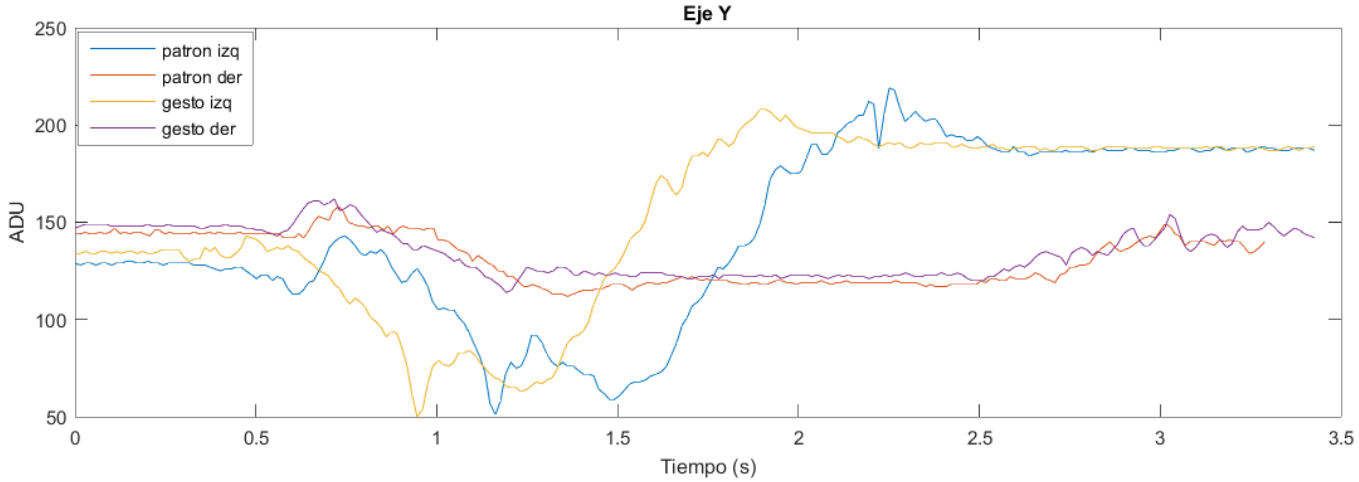


Figura 6-6. Patrones y gestos - Eje Y

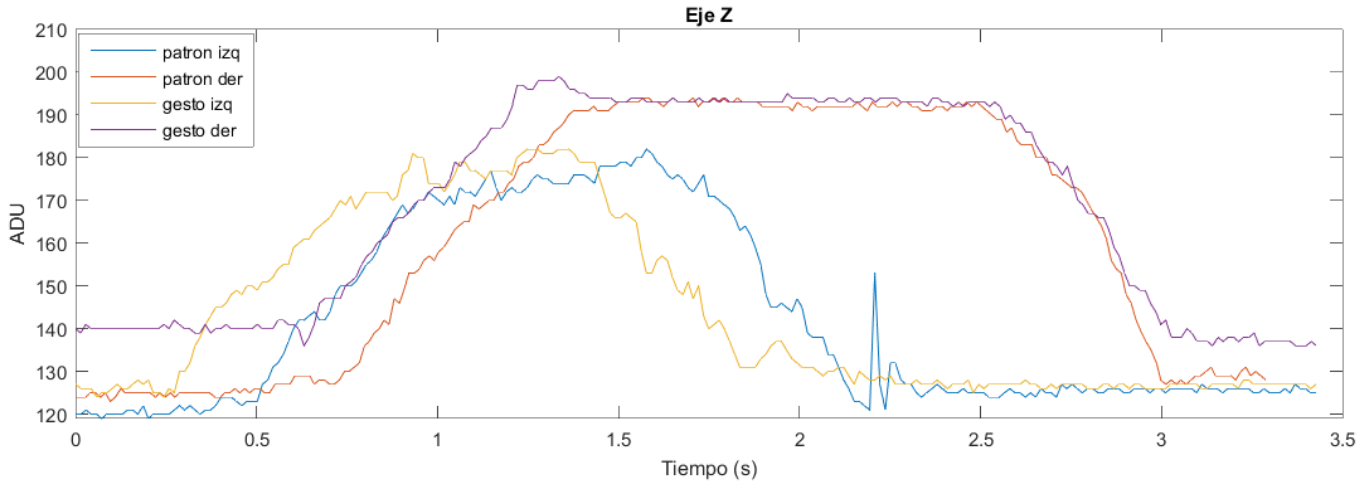


Figura 6-7. Patrones y gestos - Eje Z

Con esta información visual, es posible apreciar cómo deberían modificarse los coeficientes de correlación en cada caso.

Por poner un ejemplo, en el caso del eje Z, sobre estas líneas, se puede afirmar con rotundidad que, si bien no debe existir una gran diferencia entre ellas por la morfología de las señales, el patrón izquierdo y el gesto derecho deberían tener un coeficiente de correlación claramente inferior al del patrón izquierdo y el gesto izquierdo.

Evidentemente, a la hora de comparar si existe o no parecido entre dos señales, como se comentó en el Apartado 5, los ejes se van a comparar con sus pares, es decir, de cada gesto, el eje X se comparará con el eje X de cada uno de los dos patrones, el eje Y con el eje Y de los patrones y el Z con el eje Z de los patrones.

Esto es lo que se recoge en la siguiente tabla, y se va a calcular mediante un pequeño script para MATLAB® llamado `corr_gestos_patrones.m`. **Se encuentra incluido en los Anexos**.

Tabla 6-1. Coeficientes de correlación. Gestos VS Patrones

	Gesto Izquierdo		Gesto Derecho	
Eje	Patrón Izq	Patrón Der	Patrón Izq	Patrón Der
X	0.8486	0.5017	0.6835	0.8753
Y	0.7698	-0.3801	0.0610	0.8600
Z	0.8164	-0.0124	0.5520	0.9364

Por continuar con el ejemplo anterior, se puede observar cómo, efectivamente, el coeficiente de correlación entre el eje Z del gesto derecho y el del patrón izquierdo (0.5520) es inferior al del gesto izquierdo y el patrón izquierdo (0.8164). Sin embargo, como se observaba en la Figura 6-7, son dos señales claramente correladas, por lo que, a pesar de ser menor, el coeficiente de correlación es claramente positivo.

6.3 Montaje del prototipo y pruebas.

El montaje se ha realizado acorde a lo explicado en el apartado 4.4.

Para comenzar, se han construido los dispositivos de señalización. La disposición de los LEDs ha sido la siguiente:

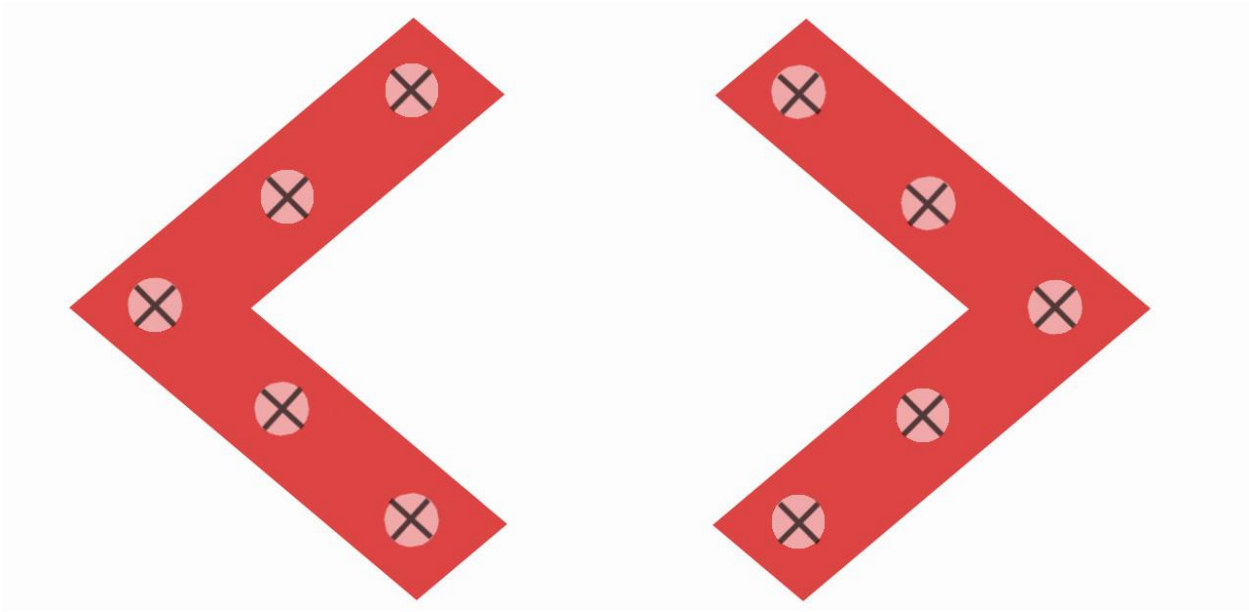


Figura 6-8. Distribución física de los LEDs en los dispositivos señaladores

Para garantizar la correcta sujeción de los diodos, ha sido necesario idear una forma de evitar que se muevan cuando el usuario de la camiseta realiza movimientos con el cuerpo. Se han utilizado dos esponjas de 5mm de altura, justo la que tienen los LEDs. Se han agujereado y se han pinchado los diodos, realizando las conexiones por detrás. De esta manera el dispositivo se mantiene flexible, adaptándose a la forma de la espalda del ciclista, cosa que no hubiera sido posible de introducir un PCB de fibra de vidrio o cualquier otro material habitual por ser completamente rígidos.

Las soldaduras para realizar las conexiones se han recubierto con una funda termorretráctil con un doble objetivo: reforzar las uniones dándoles un poco más de rigidez, y minimizar el riesgo de cortocircuitos entre los diferentes componentes. Aun así, ha habido que utilizar cinta aislante para aislar los puntos comunes de conexión, ya que las fundas retráctiles no son aplicables en esos lugares.

El esquema utilizado para la conexión de los diodos, la resistencia y los pines de conexión con FLORA, es el indicado en la Figura 4-10.

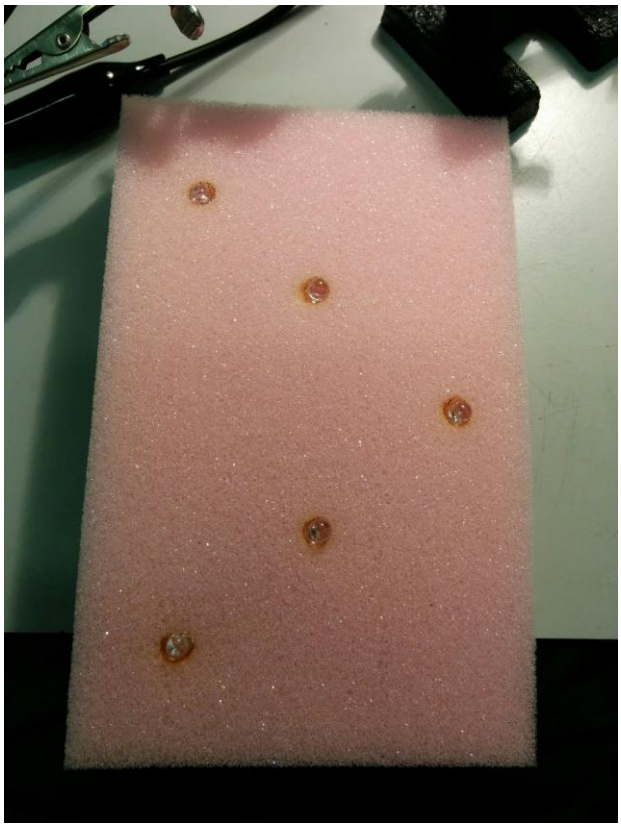


Figura 6-9. Disposición de los LEDs en la esponja

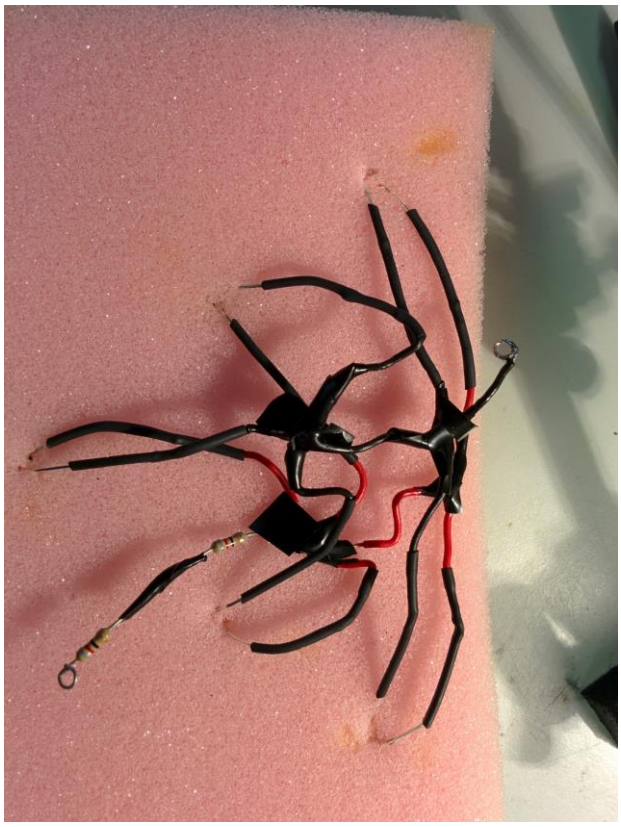


Figura 6-10. Conexionado de los LEDs

Tras realizar el conexionado, se ha comprobado que todo estaba correcto.

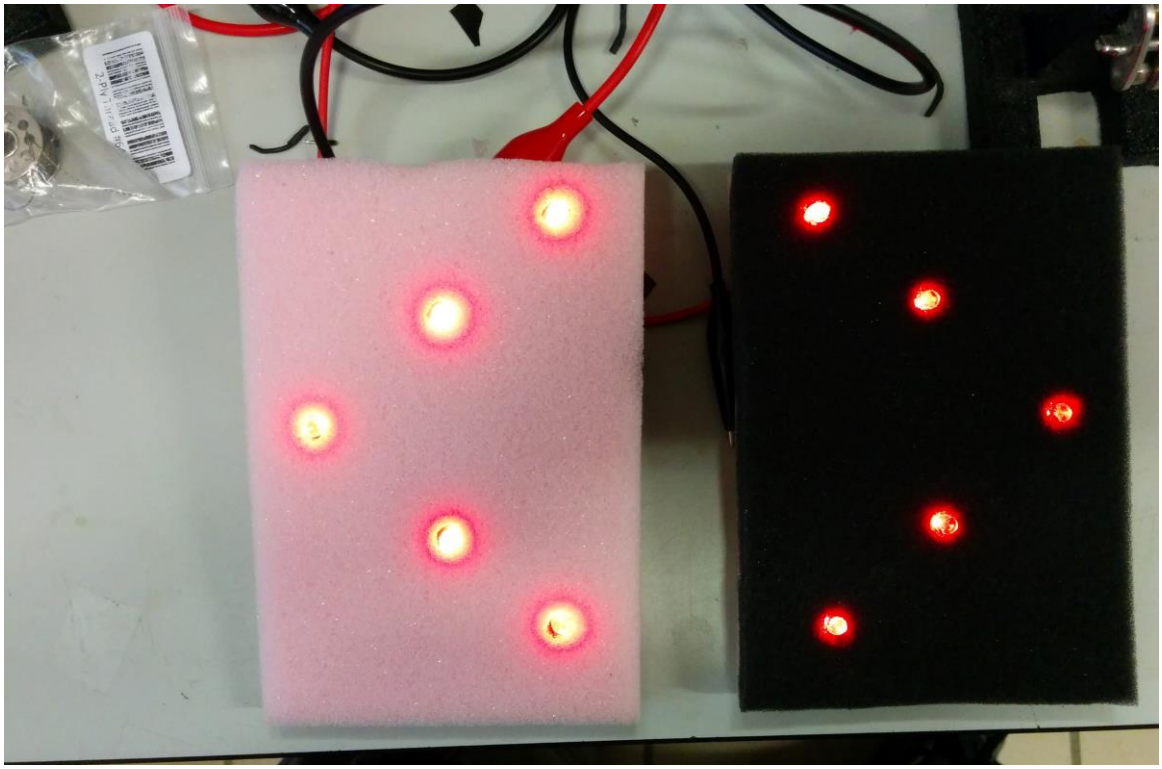


Figura 6-11. Comprobación del funcionamiento de los indicadores

Antes de pasar a la fase de unión a la camiseta, se ha hecho una serie de pruebas con diferentes tejidos. El

objetivo era utilizar el que mejor dejaba que se vieran los indicadores. Tras probar distintos tejidos de algodón, se vio que éste material era lo suficientemente denso como para que no se vieran demasiado los LEDs. El mejor resultado se obtuvo en tejido deportivo negro, que fue el finalmente escogido para el prototipo.

Figura 6-12. Comprobación de la luminosidad en distintos tejidos

Una vez que se han construído los indicadores, es momento de incorporarlos a la camiseta, no sin antes tomar medidas para tener la mejor simetría posible.

Se ha decidido colocar los LEDs, al igual que se indica en el concepto del prototipo, en la parte media-baja de la espalda. La razón es que esa es la posición más visible para el resto de conductores, ya que, en condiciones normales, el ciclista va encorvado sobre la bicicleta, por lo que los indicadores quedarían posicionados de forma casi perpendicular y en la línea de visión del conductor que va detrás en otro vehículo.

Figura 6-13. Dispositivos de señalización. Presentación y aspecto final

A continuación, es el turno de FLORA. Se ha decidido orientarla convenientemente para evitar tener que realizar las conexiones de forma extraña. Hay que recordar que el equivalente a las pistas de cobre de un PCB habitual es, en este caso, el hilo conductor, por lo que hay que evitar que se crucen las puntadas.

De esta manera, los pines superiores irán a la manga izquierda, brazo donde se situará el acelerómetro, y los pines inferiores irán hacia los dispositivos de señalización. Hay dos pines GND en la parte inferior, por lo que para facilitar el conexionado, se ha utilizado uno para cada intermitente.

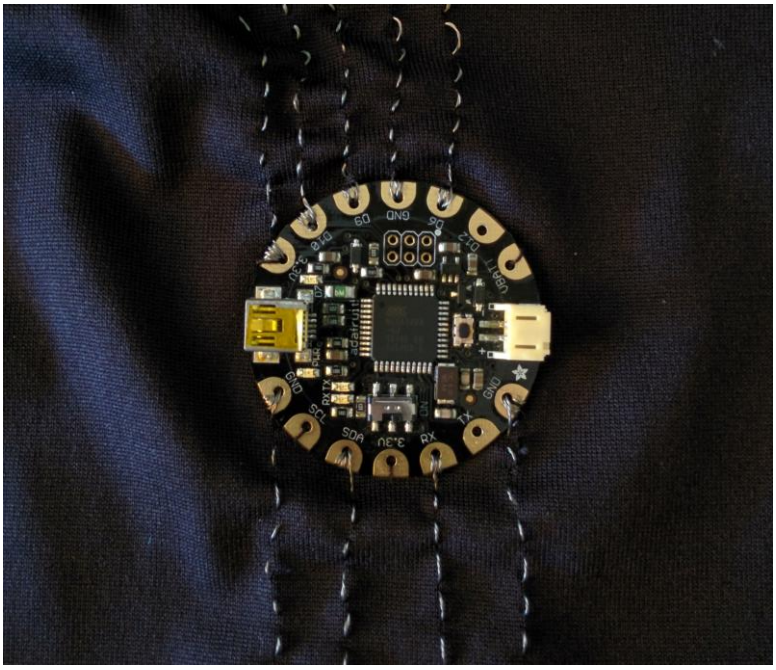


Figura 6-14. Conexionado de FLORA

En la manga, los cinco hilos que van hacia el acelerómetro se han rematado con arandelas, fabricadas a partir de los recortes sobrantes de las patas de las resistencias, que se dimensionaron para soldarlas a los LEDs de los intermitentes. Se les ha dado forma circular y se han estañado con el fin de que formen un círculo cerrado. Así, reciclando estas partes, se refuerza el lugar de conexión de los cables del acelerómetro, evitando desgarros del hilo conductor que puedan producir cortes en la alimentación o en el envío de datos desde algún eje del acelerómetro. Se puede observar en la Figura 6-15.

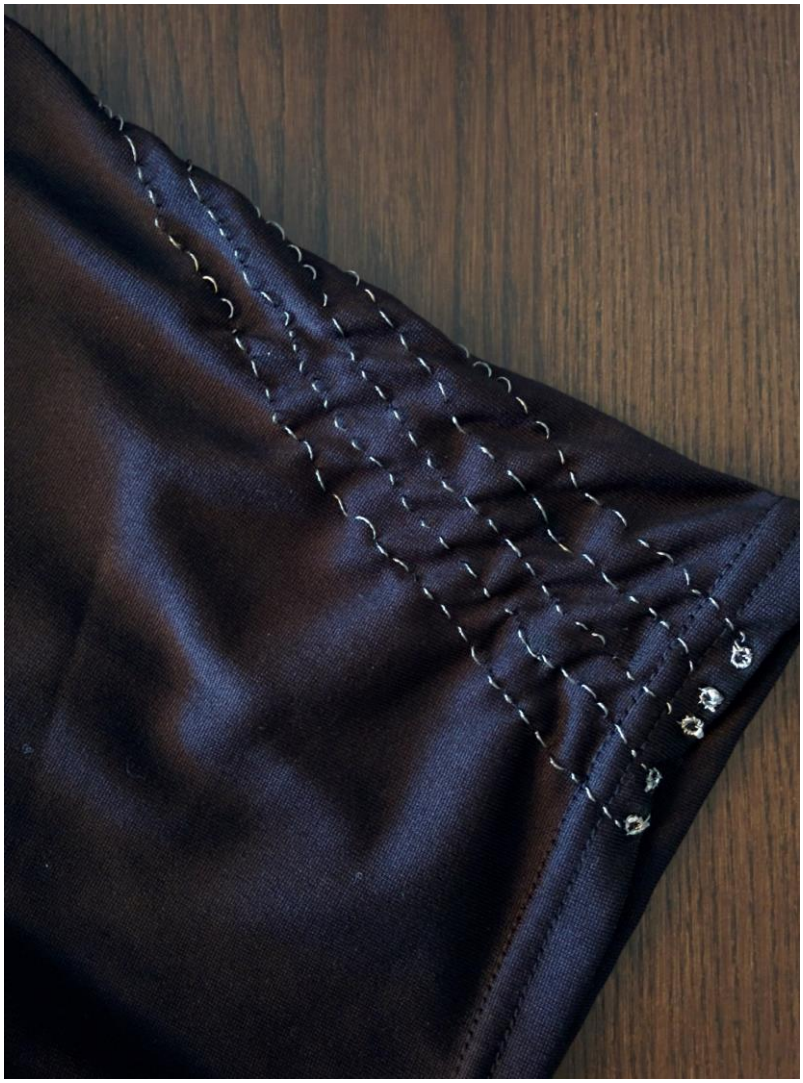


Figura 6-15. Manga izquierda. Puntos de conexión con el acelerómetro

La correspondencia de los puntos de conexión es la siguiente (de arriba a la derecha hacia abajo a la izquierda):

Tabla 6-2. Correspondencia puntos de conexión – pines de FLORA

FLORA	Acelerómetro
3.3V	VCC
D10	Eje X
D9	Eje Y
GND	GND
D6	Eje Z

Al igual que sucede con los pines superiores, los cuatro pines inferiores, que van hacia los intermitentes, se han cosido paralelamente. Gracias a su simetría, FLORA ha facilitado el conexionado de los mismos. Los hilos más externos están conectados a los pines de GND. Los dos hilos centrales, que van a las resistencias de los diodos, están conectados a los respectivos pines digitales de activación del intermitente izquierdo y derecho.

Todo esto se puede ver en la Figura 6-16.

Figura 6-16. Conexionado de los dispositivos de señalización

Con esto, ya está finalizado el prototipo.

Figura 6-17. Prototipo. Aspecto final

A la hora de realizar pruebas con el prototipo, se ha cargado directamente el Código 7 en FLORA, pero con una pequeña modificación: la función `setup` se ha cambiado para que los LEDs se enciendan desde el principio usando las funciones `digitalWrite(led_izq1,HIGH)` y `digitalWrite(led_der1,HIGH)`.

Posteriormente se restaurará el Código 7 a su funcionamiento habitual, pero encender los indicadores desde el principio facilita mucho la depuración de su funcionamiento, detectando fallos tales como soldaduras rotas al comprimir los cables para que ocupen menos espacio.

Tras comprobar que todo está correcto, se pasa a probar el prototipo al completo.

Se han hecho las pruebas con abundante luz, de forma que se refleje bien cómo se ve el dispositivo de señalización **en las peores condiciones posibles**.

Figura 6-18. Prototipo. Vista lateral

Figura 6-19. Prototipo. Vista trasera.

El acelerómetro se coloca en el brazo izquierdo, en la muñeca. La conexión con la camiseta se realiza con cables cocodrilo-cocodrilo, como se puede observar en la Figura 6-20.

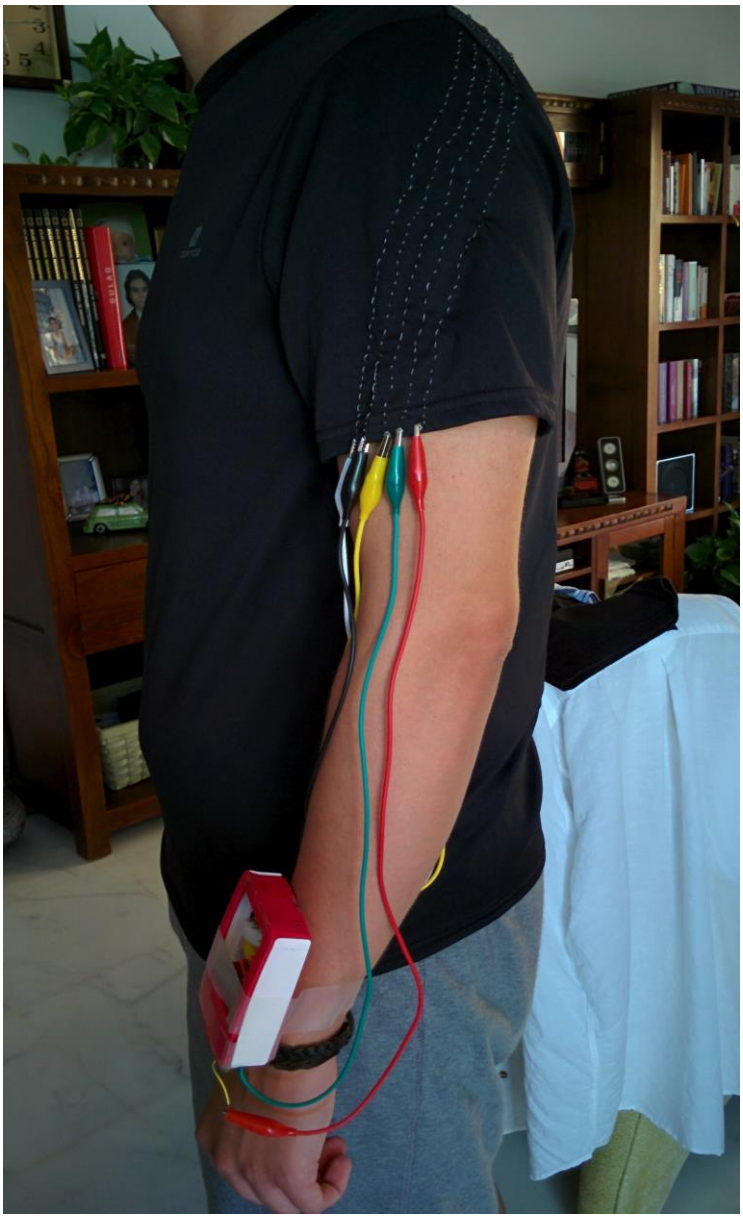


Figura 6-20. Prototipo. Conexión del acelerómetro.

Finalmente se conecta la batería a FLORA y el dispositivo se inicia. El resultado lo vemos en las Figuras 6-21 y 6-22.

El gesto derecho (Figuras 6-21) tiene dos fases: levantar el brazo con el codo haciendo un ángulo (ver Figura 1-2) y bajar el brazo hasta su posición inicial. A partir de aquí es cuando FLORA detecta que el gesto se corresponde al patrón derecho y pasa al estado de encender el intermitente derecho.

El gesto izquierdo (Figura 6-22) se compone igualmente de dos fases. La primera fase es compartida con el gesto derecho, es decir, consiste en levantar el brazo en sentido vertical con el codo haciendo un ángulo, y la segunda es la extensión completa del brazo en sentido horizontal (Figura 1-1).

Esta manera de hacer el gesto izquierdo no es, quizás, la más natural. Normalmente el gesto se suele realizar directamente extendiendo el brazo en sentido horizontal hasta su posición final. Sin embargo, se han encontrado dificultades al tratar de identificar si un gesto es acorde a un patrón capturado siguiendo esta mecánica.

Debido a la falta de movimiento en uno de los ejes (el responsable de la aceleración en sentido vertical) y a la poca aceleración que sufren los otros dos ejes a la hora de realizar este gesto, es muy complicado detectar un movimiento hecho de esta manera. De hecho, un gesto capturado acorde a esta mecánica tiene, en los dos ejes

donde se detecta aceleración apreciable, un muy bajo rango dinámico, es decir, las aceleraciones detectadas son, aún así, muy bajas. Por ello, la distinción es tan compleja, porque no siguien una tendencia concreta, las señales no tienen, morfológicamente, una forma característica sino que más bien se confunden con ruido. Es muy complicado distinguir un gesto hecho así de un ruido, por lo que el valor del coeficiente de correlación se queda a niveles muy bajos en los ejes X y Z(entre 0.25 y 0.5 en el mejor de los casos) y prácticamente a 0 en el eje Y.

Figura 6-21. Gesto de giro a la derecha

Figura 6-22. Gesto de giro a la izquierda

7 CONCLUSIONES Y TRABAJO FUTURO

A lo largo de esta memoria se ha explicado la creación de un sistema electrónico textil para señalización ciclista mediante reconocimiento de gestos, desde el concepto o idea inicial hasta su implementación física. También se ha desarrollado un algoritmo de reconocimiento de gestos basado en la utilización de un acelerómetro analógico MMA7260Q y una placa de desarrollo Adafruit FLORA.

La idea general ha sido crear una camiseta inteligente que, de forma autónoma, detecte los gestos realizados con el brazo y compruebe si responden a un patrón conocido o no, indicando, si fuese el caso, la dirección del giro que desea realizar el ciclista, actuando sobre un sistema de señalización basado en LEDs, que hacen las veces de intermitentes.

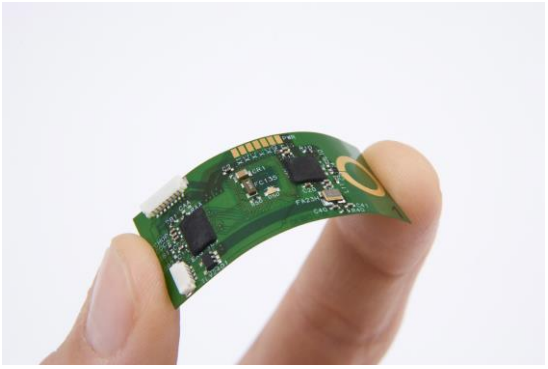
El corazón del desarrollo del proyecto ha sido el algoritmo de reconocimiento de gestos. En la actualidad, existen una gran cantidad de sistemas para ciclistas basados en pulsadores y botones que, en este caso, se han pretendido sustituir por el brazo del ciclista, gestionando los dispositivos de señalización acorde a los gestos realizados, todo ello siguiendo las recomendaciones de la Dirección General de Tráfico.

Este proyecto se puede tomar como base, continuando su desarrollo para hacerlo más completo y funcional.

A continuación, se propone una serie de modificaciones que pueden añadir gran cantidad de funcionalidades al dispositivo, así como mejorar lo ya implementado.

7.1 Miniaturización del acelerómetro

Una de las modificaciones más inmediatas que saltan a la vista nada más ver el prototipo es la necesidad de miniaturizar el acelerómetro, incorporándolo en algún tipo de pulsera, reloj o muñequera.



Sería muy interesante ver cómo se podría trabajar con los acelerómetros incorporados en los podómetros de pulsera o en pulseras biométricas, que disponen de una alta precisión. Intentar extraer esos datos del acelerómetro y trasmitirlos a FLORA podría solucionar el problema de la miniaturización del acelerómetro, no siendo necesario, por tanto, construir un nuevo dispositivo.

En caso de miniaturizar el MMA7260Q, habría que valorar realizarlo en PCB flexible, ya que de esta forma se puede adaptar mucho mejor a la forma cilíndrica de una pulsera.

7.2 Inclusión de interfaz radio

En este Proyecto se han utilizado cables de test para conectar el acelerómetro con FLORA.

En una primera etapa, se planteó utilizar Bluetooth para transmitir los datos, eliminando la necesidad de rutar cables hasta el acelerómetro.

Se hicieron pruebas con un módulo Bluetooth HC-05, pero el alto consumo de este último para un dispositivo wearable (50mA) y la necesidad de la incorporación de interfaz radio en FLORA, hicieron que se descartara esta posibilidad.

Al igual que se comentaba en el apartado anterior, hay pulseras de monitorización de actividad que disponen de una API para desarrollar aplicaciones móviles. Estas pulseras, normalmente envían la información a un Smartphone, donde se realizan las estadísticas, pero por sí mismas no suelen almacenar gran cantidad de información.

Si se le añade interfaz radio (Bluetooth LE, o Bluetooth 5.0, que está previsto que llegue en 2016-17) a FLORA, habría que estudiar la posibilidad de sincronizar FLORA con esta pulsera, utilizando su acelerómetro y su interfaz radio para transmitir la información de nuestro brazo. De esta forma, se eliminan dos necesidades: la de miniaturizar el acelerómetro y la de transmisión de información de manera inalámbrica. Además, hace que no haya que rutar el hilo conductor hasta la manga de la camiseta.

7.3 Información y rutas

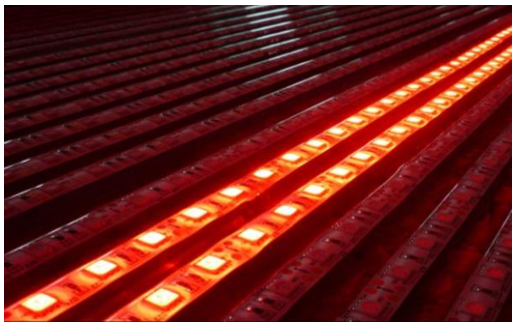
Si en lugar de con una pulsera de monitorización de actividad física, se lograse sincronizar el sistema de señalización con un Smart-watch vía APP, cabría la posibilidad de enviar información del tráfico y rutas alternativas a los usuarios que quieran ir a un destino concreto a través de aplicaciones como OpenMaps, Google Maps o Mapas de Apple.

Estos relojes inteligentes también disponen de acelerómetro e interfaz radio, por lo que lo descrito en los dos apartados anteriores sería perfectamente implementable.

7.4 Mejora del dispositivo de señalización

Algo inmediatamente mejorable de este sistema es el dispositivo de señalización.

Para el prototipo se han utilizado LEDs de alta luminosidad de montaje de orificio pasante (THD). Actualmente, la popularidad de sistemas de iluminación LED ha permitido una importante bajada en los costes de las tiras de LEDs de montaje superficial (SMD).



Estas tiras tienen una gran integración de LEDs, entre 3 y 6 leds cada 5cm, por lo que el sistema mejoraría mucho en luminosidad, además de tener un perfil más bajo, evitando la necesidad de utilizar medios para mantener físicamente la posición de los LEDs, ya que irían directamente puestos sobre la camiseta y cubiertos con algún medio plástico que hiciera las veces de "pantalla".

7.5 Reconocimiento de gestos

Por último, para finalizar con este apartado y con la memoria de este proyecto, se propone una mejora en el sistema de reconocimiento de gestos.

Se puede incrementar la seguridad a la hora de detectar un gesto como correcto, minimizando la probabilidad de que se dé un falso positivo. Esto se puede de varias formas:

- Software:
    - Poniendo condiciones para que, en caso de un gesto "neutro", es decir, que se dé el caso de que el gesto tenga un parecido suficiente para dar positivo en su identificación tanto con el patrón derecho como con el izquierdo, se decida optar por el de mayor parecido. Es un caso muy poco probable, pero por si se da, sería inteligente blindar el sistema para que no incurra en el error.

    - En el código, primero se comprueba si se cumplen las características del patrón derecho y, en caso negativo, se comprueba si se cumplen las del izquierdo. Esto da cierta ventaja al patrón derecho sobre el izquierdo, y puede acarrear un falso positivo si se cumplen las condiciones de gesto "neutro", ya que, en ese hipotético caso, se tomaría como un gesto para girar a la derecha, sin comprobar si el parecido con el patrón izquierdo es mayor.

- Hardware:
    - Mediante el uso de un segundo acelerómetro situado en la zona superior del húmero, cerca del hombro. Se guardarían los datos correspondientes a los gestos de derecha e izquierda a modo de patrones, igual que se hace con el acelerómetro de la muñeca, y se produciría un doble contraste, siendo necesario un positivo en la muñeca y un positivo en el hombro (para el mismo sentido del giro) para tomar un gesto como una indicación de giro válida. De esta forma sería mucho más complicado incurrir en un error.
    - Los ciclistas no tienen por qué llevar pulsera, pero sí suelen llevar guantes. Sería interesante ver cómo se podría reubicar el acelerómetro, pasando de la muñeca al guante. En el guante las aceleraciones serían mayores y los gestos más definidos aún. De cara al reconocimiento de gestos, debería estudiarse cómo afectan las vibraciones de la bicicleta, que serán siempre mayores en el guante respecto a las que puedan darse en la muñeca, pero podría ser una manera de ampliar el rango dinámico en ejes como el Y del patrón izquierdo, que es menos definido que, por ejemplo, el eje X del patrón derecho.

ANEXO A. CÓDIGOS DE ARDUINO

A.1 Código 1. Captura de un gesto.

captura_gesto.ino

```
unsigned long time;
char rxChar;


void setup() {
  // Inicializacion de comunicación serie a 115200 bits por segundo:
  Serial.begin(115200);
  pinMode(A10, INPUT);
  pinMode(A9, INPUT);
  pinMode(A7, INPUT);

}

void loop() {

 if(Serial.available())
 {
   rxChar = Serial.read();
   if(rxChar=='1')
   {
     while(rxChar != '0'){

     int sensorXValue = analogRead(A10);
     Serial.println(sensorXValue);
     int sensorYValue = analogRead(A9);
     Serial.println(sensorYValue);
     int sensorZValue = analogRead(A7);
     time=millis();
     Serial.println(sensorZValue);
     Serial.println(time);
     if(Serial.available()){
        rxChar=Serial.read();
     }
   }
  }
 }
delay(16);
}
```

A.2 Código 3. Correlación cruzada en FLORA.

cc_arduino.ino

```
#include <avr/pgmspace.h>
#include <stdint.h>
#include <math.h>


//PATRÓN DERECHO
const uint16_t GiroDerX[] PROGMEM = {85, 86, 85, 86, 85, 86, 86, 86, 83, 87, 87, 87, 85,
84, 84,
              87, 85, 85, 86, 86, 85, 85, 85, 85, 86, 85, 86, 85, 85, 86,
              85, 86, 85, 85, 86, 86, 86, 84, 86, 85, 84, 84, 81, 81, 77,
              72, 67, 64, 62, 66, 69, 70, 70, 72, 72, 72, 73, 76, 78, 80,
              82, 84, 86, 88, 89, 87, 87, 89, 91, 95, 96, 93, 92, 96, 98,
              99, 99, 101, 105, 103, 105, 107, 111, 111, 112, 113, 112,
```

```
              116, 119, 122, 123, 122, 124, 128, 126, 126, 128, 131, 131,
              131, 131, 132, 133, 134, 133, 134, 133, 135, 136, 136, 137,
              139, 138, 139, 140, 141, 140, 140, 140, 139, 138, 137, 137,
              136, 135, 136, 134, 133, 133, 133, 132, 132, 133, 132, 133,
              134, 134, 134, 135, 133, 133, 134, 133, 133, 132, 132, 133,
              134, 134, 133, 134, 134, 134, 134, 134, 133, 134, 134, 132,
              133, 132, 133, 132, 133, 133, 132, 134, 133, 134, 134, 134,
              134, 134, 135, 136, 135, 135, 134, 135, 136, 137, 138, 138,
              139, 139, 138, 138, 136, 135, 133, 131, 128, 127, 124, 124,
              121, 120, 112, 112, 111, 108, 107, 107, 107, 109, 110, 110,
              105, 101, 97, 95, 92, 90, 87, 82, 77, 77, 76, 73, 70, 70,
              70, 68, 67, 68, 69, 70, 74, 74, 72, 73, 74, 74, 78, 77, 79,
              79, 82, 84, 86};
const uint16_t GiroDerY[] PROGMEM = {144, 144, 145, 144, 144, 145, 144, 145, 147, 144, 145,
144,
              145, 145, 144, 143, 146, 146, 144, 145, 144, 144, 144, 144,
              145, 144, 144, 145, 144, 145, 144, 144, 144, 145, 144, 144,
              144, 144, 144, 144, 144, 144, 142, 142, 142, 144, 142, 146,
              150, 153, 152, 151, 156, 158, 155, 150, 149, 148, 148, 147,
              148, 148, 146, 148, 144, 147, 148, 147, 147, 147, 146, 147,
              147, 141, 141, 140, 138, 136, 135, 134, 130, 133, 131, 129,
              128, 125, 125, 122, 122, 120, 117, 117, 118, 117, 116, 115,
              113, 113, 113, 112, 113, 114, 115, 115, 115, 116, 116, 117,
              118, 118, 118, 117, 115, 117, 118, 119, 119, 118, 119, 119,
              120, 121, 122, 121, 122, 121, 120, 121, 120, 120, 120, 120,
              119, 119, 118, 118, 119, 119, 118, 118, 119, 120, 120, 119,
              119, 118, 119, 119, 119, 119, 120, 119, 118, 119, 119, 120,
              119, 119, 119, 119, 119, 119, 119, 119, 119, 120, 119, 119,
              119, 119, 118, 117, 118, 117, 117, 117, 118, 118, 118, 118,
              118, 118, 120, 119, 121, 121, 120, 120, 122, 121, 121, 121,
              122, 124, 123, 121, 120, 119, 123, 124, 127, 127, 128, 128,
              129, 133, 136, 138, 140, 137, 135, 136, 137, 139, 141, 142,
              143, 142, 145, 149, 148, 144, 143, 138, 140, 140, 140, 140,
              139, 138, 141, 140, 141, 140, 140, 137, 134, 135, 137, 140};
const uint16_t GiroDerZ[] PROGMEM = {124, 124, 124, 125, 125, 124, 125, 123, 124, 126, 125,
125,
              125, 125, 125, 125, 124, 125, 124, 125, 124, 125, 125, 125,
              125, 125, 125, 125, 124, 124, 125, 125, 126, 125, 126, 125,
              126, 126, 125, 125, 127, 127, 127, 127, 129, 129, 129, 129,
              127, 128, 128, 127, 127, 128, 130, 130, 131, 132, 136, 137,
              139, 140, 142, 141, 147, 146, 149, 153, 153, 154, 156, 157,
              156, 158, 159, 161, 163, 164, 165, 165, 169, 168, 169, 170,
              170, 172, 172, 174, 175, 178, 179, 179, 180, 183, 183, 184,
              186, 187, 188, 190, 191, 191, 191, 191, 192, 191, 191, 191,
              192, 193, 193, 193, 193, 193, 194, 194, 193, 193, 192, 193,
              194, 193, 193, 193, 194, 192, 193, 193, 194, 193, 194, 193,
              193, 194, 193, 193, 193, 193, 192, 192, 192, 192, 192, 192,
              191, 193, 192, 192, 191, 192, 192, 192, 192, 192, 192, 192,
              192, 193, 192, 191, 193, 192, 192, 192, 193, 192, 193, 193,
              192, 192, 191, 191, 191, 192, 192, 191, 192, 192, 192, 193,
              192, 193, 192, 191, 190, 189, 189, 186, 187, 184, 183, 183,
              183, 180, 180, 180, 176, 176, 175, 174, 173, 173, 172, 170,
              168, 166, 164, 161, 156, 154, 153, 148, 146, 142, 139, 137,
              135, 132, 128, 127, 128, 127, 128, 127, 128, 129, 129, 130,
              131, 129, 129, 129, 129, 128, 130, 131, 129, 130, 129, 128};

//GESTO PRUEBA GIRO DERECHA
const uint16_t gestoX[] PROGMEM = {86, 86, 87, 86, 88, 88, 88, 87, 87, 87, 86, 87, 86, 86,
87,
             87, 87, 87, 87, 87, 87, 88, 87, 87, 87, 88, 87, 87, 87, 87,
             88, 87, 86, 85, 86, 85, 83, 80, 76, 75, 72, 72, 71, 74, 74,
             72, 71, 71, 71, 74, 76, 80, 85, 87, 90, 93, 96, 97, 102, 105,
             106, 109, 109, 110, 112, 115, 116, 114, 115, 115, 115, 117,
             120, 123, 123, 125, 126, 128, 131, 132, 132, 134, 135, 138,
             139, 141, 139, 138, 140, 141, 142, 141, 141, 141, 141, 141,
             140, 139, 138, 139, 138, 138, 139, 139, 139, 139, 139, 139,
             139, 140, 139, 138, 138, 138, 137, 136, 136, 136, 135, 135,
             135, 134, 136, 136, 137, 135, 137, 137, 137, 137, 137, 137,
             136, 136, 137, 137, 137, 137, 137, 137, 138, 139, 138, 138,
             138, 137, 137, 137, 137, 136, 138, 137, 138, 137, 138, 137,
```

```
              139, 138, 139, 138, 139, 139, 139, 140, 140, 142, 142, 141,
              143, 143, 143, 143, 143, 141, 142, 139, 137, 136, 133, 132,
              131, 128, 129, 129, 129, 126, 124, 122, 120, 121, 120, 117,
              118, 114, 107, 105, 103, 104, 105, 103, 98, 95, 90, 85, 80,
              79, 76, 75, 75, 69, 66, 60, 61, 62, 65, 70, 75, 78, 80, 82,
              85, 87, 88, 89, 88, 87, 85, 85, 83, 81, 82, 83, 86, 87, 86,
              87, 86, 85, 84, 83};
const uint16_t gestoY[] PROGMEM = {147, 148, 149, 149, 149, 149, 149, 148, 148, 148, 148,
148,
              148, 148, 149, 148, 148, 148, 149, 148, 148, 148, 148, 148,
              147, 147, 148, 148, 148, 149, 148, 148, 148, 147, 147, 146,
              146, 145, 144, 143, 145, 146, 149, 153, 157, 160, 161, 161,
              159, 160, 162, 156, 157, 159, 158, 154, 151, 148, 145, 146,
              145, 144, 142, 139, 139, 136, 136, 138, 137, 136, 135, 134,
              132, 130, 131, 128, 127, 127, 125, 122, 121, 119, 117, 114,
              115, 119, 123, 127, 126, 125, 125, 124, 125, 127, 127, 126,
              123, 123, 125, 124, 123, 124, 123, 123, 124, 123, 123, 122,
              122, 124, 124, 124, 124, 124, 123, 123, 122, 122, 121, 121,
              122, 122, 122, 123, 121, 123, 123, 123, 123, 122, 122, 122,
              123, 122, 122, 123, 121, 122, 123, 123, 123, 123, 122, 123,
              122, 121, 122, 123, 122, 123, 122, 122, 122, 121, 122, 122,
              123, 122, 123, 123, 124, 123, 122, 122, 123, 123, 123, 124,
              123, 123, 124, 123, 121, 120, 120, 120, 122, 122, 124, 126,
              125, 127, 128, 127, 129, 132, 134, 135, 134, 133, 132, 128,
              134, 136, 137, 136, 133, 135, 134, 133, 136, 139, 144, 146,
              147, 141, 138, 138, 141, 144, 147, 154, 152, 144, 137, 135,
              137, 141, 143, 144, 143, 140, 137, 139, 145, 148, 146, 146,
              146, 147, 150, 148, 145, 143, 145, 147, 146, 144, 143, 142};
const uint16_t gestoZ[] PROGMEM = {140, 139, 141, 140, 140, 140, 140, 140, 140, 140, 140,
140,
              140, 140, 140, 140, 140, 141, 140, 142, 141, 140, 140, 139,
              139, 141, 139, 140, 140, 141, 140, 140, 140, 140, 140, 141,
              140, 140, 140, 142, 142, 141, 141, 140, 136, 138, 141, 146,
              147, 147, 147, 147, 150, 151, 152, 155, 157, 158, 160, 161,
              162, 165, 166, 166, 167, 169, 170, 170, 172, 173, 173, 173,
              175, 179, 178, 180, 181, 182, 184, 185, 187, 187, 187, 189,
              192, 197, 197, 196, 196, 198, 198, 198, 198, 199, 198, 196,
              196, 195, 195, 194, 194, 194, 194, 194, 193, 193, 193, 194,
              194, 194, 193, 193, 193, 193, 193, 194, 193, 193, 193, 193,
              194, 193, 193, 194, 193, 194, 193, 193, 193, 193, 193, 193,
              193, 193, 193, 193, 193, 195, 194, 194, 194, 194, 194, 193,
              193, 194, 194, 193, 193, 194, 193, 193, 194, 194, 193, 194,
              194, 194, 194, 193, 193, 193, 193, 193, 192, 194, 194, 193,
              193, 192, 193, 192, 193, 193, 193, 193, 192, 193, 192, 189,
              190, 188, 188, 186, 186, 183, 181, 179, 179, 178, 176, 178,
              174, 170, 169, 167, 167, 166, 166, 163, 159, 157, 153, 150,
              150, 149, 149, 146, 144, 141, 142, 138, 138, 138, 140, 139,
              137, 137, 137, 136, 138, 137, 138, 138, 137, 138, 138, 139,
              136, 137, 137, 137, 137, 137, 137, 136, 136, 136, 137, 136};

int i = 0;

int n = 240;
int READY = 2;
int END = 3;
char rxChar;
 double varX=0;
 double varY=0;
 double varZ=0;
 double patronX=0;
 double patronY=0;
 double patronZ=0;
 double Xxy=0;
 double Yxy=0;
 double Zxy=0;
 double Xxsquare=0;
 double Yxsquare=0;
 double Zxsquare=0;
 double Xysquare=0;
```

```
 double Yysquare=0;
 double Zysquare=0;
 double Xxsum=0;
 double Yxsum=0;
 double Zxsum=0;
 double Xysum=0;
 double Yysum=0;
 double Zysum = 0;
 double Xxysum=0;
 double Yxysum=0;
 double Zxysum=0;
 double Xxsqr_sum=0;
 double Yxsqr_sum=0;
 double Zxsqr_sum=0;
 double Xysqr_sum=0;
 double Yysqr_sum=0;
 double Zysqr_sum=0;
 double numX=0;
 double denoX=0;
 double numY=0;
 double denoY=0;
 double numZ=0;
 double denoZ=0;
 double coeffX=0;
 double coeffY=0;
 double coeffZ=0;


void setup() {
  // Inicializar comunicación por puerto serie a 115200 bits por segundo
  Serial.begin(115200);
  pinMode(A10, INPUT);
  pinMode(A9, INPUT);
  pinMode(A7, INPUT);
}

void loop() {
  if(Serial.available())
     {
     rxChar = Serial.read();
     if(rxChar=='1')
     {
        for (i=0; i<n; i++)
        {

         //Lectura del patrón

          patronX=pgm_read_word_near(GiroDerX + i);
          patronY=pgm_read_word_near(GiroDerY + i);
          patronZ=pgm_read_word_near(GiroDerZ + i);

          varX=pgm_read_word_near(gestoX + i);
          varY=pgm_read_word_near(gestoY + i);
          varZ=pgm_read_word_near(gestoZ + i);


          //Correlacion cruzada eje X
          Xxy= varX * patronX;
          Xxsquare = varX * varX;
          Xysquare = patronX * patronX;
          Xxsum = Xxsum + varX;
          Xysum = Xysum + patronX;
          Xxysum = Xxysum + Xxy;
          Xxsqr_sum = Xxsqr_sum + Xxsquare;
          Xysqr_sum = Xysqr_sum + Xysquare;
```

```
          //Correlacion cruzada eje Y

          Yxy= varY * patronY;
          Yxsquare = varY * varY;
          Yysquare = patronY * patronY;
          Yxsum = Yxsum + varY;
          Yysum = Yysum + patronY;
          Yxysum = Yxysum + Yxy;
          Yxsqr_sum = Yxsqr_sum + Yxsquare;
          Yysqr_sum = Yysqr_sum + Yysquare;

          //Correlacion cruzada eje Z

          Zxy = varZ * patronZ;
          Zxsquare = varZ * varZ;
          Zysquare = patronZ * patronZ;
          Zxsum = Zxsum + varZ;
          Zysum = Zysum + patronZ;
          Zxysum = Zxysum + Zxy;
          Zxsqr_sum = Zxsqr_sum + Zxsquare;
          Zysqr_sum = Zysqr_sum + Zysquare;

        }

        //Coeficiente Correlación Cruzada X: GESTO GIRO DERECHA
        numX=1.0*((n*Xxysum)-(Xxsum*Xysum));
        denoX=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
        coeffX=numX/sqrt(denoX);


        //Coeficiente Correlacion Cruzada Y: GESTO GIRO DERECHA
        numY=1.0*((n*Yxysum)-(Yxsum*Yysum));
        denoY=1.0*((n*Yxsqr_sum-Yxsum*Yxsum)*(n*Yysqr_sum-Yysum*Yysum));
        coeffY=numY/sqrt(denoY);


        //Coeficiente Correlacion Cruzada Z: GESTO GIRO DERECHA
        numZ=1.0*((n*Zxysum)-(Zxsum*Zysum));
        denoZ=1.0*((n*Zxsqr_sum-Zxsum*Zxsum)*(n*Zysqr_sum-Zysum*Zysum));
        coeffZ=numZ/sqrt(denoZ);


        Serial.println(READY);
        Serial.println(coeffX);
        Serial.println(coeffY);
        Serial.println(coeffZ);
        Serial.println(END);

      }
    }

  delay(80);

}
```

cc_arduino_izq.ino

```
#include <avr/pgmspace.h>
#include <stdint.h>
#include <math.h>

//PATRÓN IZQUIERDO
const uint16_t patronIzqX[] PROGMEM = {83, 83, 84, 83, 84, 83, 84, 83, 83, 83, 83, 83, 82,
83, 82,
              82, 81, 83, 83, 82, 81, 82, 81, 80, 78, 79, 78, 77, 76, 75,
              75, 72, 70, 70, 68, 66, 66, 66, 66, 65, 64, 61, 60, 61, 63,
              67, 68, 69, 71, 74, 77, 78, 85, 91, 90, 92, 96, 102, 109,
              113, 113, 113, 113, 116, 118, 124, 126, 128, 130, 131, 132,
              134, 139, 138, 143, 145, 151, 156, 162, 167, 167, 164, 155,
              150, 148, 149, 151, 153, 151, 149, 147, 147, 149, 148, 148,
              150, 150, 151, 152, 153, 154, 156, 159, 163, 162, 161, 160,
              157, 161, 159, 157, 157, 155, 149, 145, 140, 138, 137, 135,
              132, 131, 131, 130, 131, 133, 134, 131, 131, 131, 131, 131,
              131, 131, 131, 136, 139, 140, 137, 134, 136, 137, 138, 138,
              141, 140, 141, 143, 141, 140, 141, 141, 138, 140, 150, 129,
              136, 132, 131, 136, 135, 136, 135, 134, 135, 133, 134, 134,
              134, 134, 135, 135, 135, 135, 137, 137, 135, 135, 134, 136,
              135, 135, 136, 135, 136, 136, 135, 134, 134, 134, 134, 134,
              134, 134, 134, 134, 134, 135, 134, 134, 134, 135, 134, 133,
              135, 133, 133, 133, 134, 134, 133, 133, 134, 133, 133, 133,
              133, 133, 133, 132, 134, 132, 134, 133, 133, 133, 132, 133,
              133, 133, 133, 133, 134, 133, 133, 133, 133, 134, 134, 134,
              134};
const uint16_t patronIzqY[] PROGMEM = {129, 128, 129, 129, 128, 129, 129, 128, 129, 129,
130, 130,
              129, 129, 130, 129, 129, 128, 129, 129, 129, 129, 129, 128,
              128, 128, 127, 126, 125, 126, 126, 127, 127, 125, 123, 121,
              123, 123, 120, 122, 121, 116, 113, 113, 115, 119, 120, 125,
              130, 137, 140, 142, 143, 141, 138, 134, 133, 135, 134, 136,
              133, 127, 123, 119, 120, 124, 126, 122, 117, 109, 105, 106,
              105, 105, 101, 98, 93, 87, 81, 73, 57, 51, 58, 72, 78, 75,
              76, 82, 92, 92, 88, 82, 78, 76, 78, 76, 76, 74, 72, 72, 71,
              64, 62, 59, 59, 61, 64, 67, 68, 68, 69, 71, 72, 73, 76, 82,
              88, 97, 101, 107, 109, 112, 117, 122, 127, 126, 129, 133,
              138, 138, 139, 142, 149, 159, 172, 177, 179, 177, 175, 175,
              176, 183, 190, 190, 185, 185, 188, 196, 198, 201, 202, 205,
              205, 212, 211, 188, 206, 219, 218, 209, 202, 204, 207, 204,
              202, 203, 203, 199, 194, 195, 194, 194, 192, 192, 194, 192,
              189, 188, 187, 186, 186, 189, 186, 186, 184, 185, 186, 186,
              186, 186, 186, 187, 186, 186, 186, 187, 186, 187, 188, 187,
              187, 187, 187, 188, 188, 187, 187, 187, 186, 186, 186, 187,
              187, 188, 188, 188, 187, 186, 187, 187, 189, 188, 187, 188,
              188, 188, 186, 187, 188, 189, 188, 188, 187, 187, 187, 188,
              188, 188, 188, 187};
const uint16_t patronIzqZ[] PROGMEM = {120, 120, 121, 120, 120, 119, 120, 120, 120, 120,
121, 121,
              120, 122, 119, 120, 120, 120, 120, 121, 122, 121, 122, 121,
              120, 121, 121, 122, 124, 124, 124, 123, 122, 123, 123, 123,
              127, 128, 131, 132, 134, 137, 140, 142, 142, 143, 144, 142,
              142, 144, 148, 150, 150, 150, 151, 153, 155, 156, 158, 161,
              163, 165, 167, 169, 167, 168, 170, 170, 172, 171, 170, 169,
              171, 169, 173, 172, 172, 171, 173, 175, 177, 173, 170, 172,
              173, 172, 172, 173, 175, 176, 175, 175, 174, 174, 174, 174,
              176, 176, 176, 175, 174, 178, 178, 178, 178, 179, 179, 178,
              180, 180, 182, 181, 179, 178, 176, 175, 176, 175, 173, 172,
              174, 176, 171, 171, 170, 169, 168, 166, 163, 164, 162, 159,
              155, 148, 145, 145, 146, 145, 144, 147, 145, 139, 138, 138,
              138, 134, 134, 131, 128, 126, 123, 123, 122, 121, 153, 127,
              121, 132, 132, 128, 127, 127, 125, 124, 125, 126, 127, 126,
              125, 125, 125, 126, 125, 125, 125, 125, 124, 124, 124, 125,
              126, 125, 125, 124, 125, 124, 126, 125, 125, 124, 127, 126,
              127, 126, 125, 126, 126, 125, 125, 126, 126, 126, 125, 125,
              126, 126, 126, 125, 126, 126, 125, 126, 126, 126, 126, 126,
              125, 127, 126, 126, 126, 125, 126, 126, 126, 126, 126, 125,
```

```
              125, 126, 125, 125, 125, 126, 125, 127, 126, 126, 125, 125};

//GESTO_PRUEBA_GIRO_IZQUIERDA
const uint16_t gestoX[] PROGMEM = {83, 84, 84, 84, 83, 82, 82, 80, 81, 80, 79, 79, 79, 78,
75,
             73, 71, 69, 66, 64, 62, 61, 61, 61, 63, 66, 68, 68, 70, 70,
             70, 73, 77, 81, 84, 86, 87, 88, 92, 96, 100, 102, 106, 107,
             108, 109, 111, 111, 113, 115, 117, 119, 119, 121, 122, 124,
             127, 128, 130, 130, 132, 132, 136, 140, 150, 161, 166, 162,
             151, 144, 140, 140, 142, 146, 147, 146, 147, 146, 146, 147,
             145, 145, 142, 145, 145, 147, 147, 148, 148, 145, 143, 141,
             139, 137, 138, 138, 138, 138, 138, 139, 139, 136, 135, 136,
             137, 138, 138, 138, 135, 133, 134, 142, 146, 149, 147, 141,
             141, 145, 144, 146, 147, 143, 143, 145, 145, 146, 146, 146,
             144, 144, 145, 146, 148, 148, 146, 145, 144, 144, 142, 144,
             142, 140, 140, 139, 140, 139, 140, 141, 142, 142, 141, 142,
             140, 140, 138, 141, 143, 143, 143, 140, 140, 139, 140, 139,
             140, 140, 140, 141, 141, 140, 140, 139, 140, 140, 141, 141,
             141, 141, 140, 140, 139, 140, 140, 141, 141, 140, 140, 140,
             139, 140, 140, 140, 140, 140, 139, 139, 138, 138, 139, 139,
             140, 139, 139, 138, 138, 138, 137, 138, 138, 139, 139, 139,
             138, 139, 139, 139, 139, 139, 139, 140, 139, 139, 138, 138,
             138, 139, 139, 139, 138, 138, 138, 139, 139, 139, 139, 138,
             138, 139, 139, 139};
const uint16_t gestoY[] PROGMEM = {134, 134, 135, 134, 134, 135, 135, 134, 135, 134, 135,
135,
             134, 135, 134, 134, 135, 136, 136, 136, 136, 132, 130, 131,
             131, 137, 135, 137, 133, 132, 132, 134, 137, 143, 142, 141,
             139, 135, 136, 137, 136, 138, 136, 135, 132, 130, 128, 125,
             124, 122, 118, 116, 111, 108, 111, 109, 106, 101, 99, 96,
             91, 94, 93, 86, 76, 61, 50, 54, 67, 76, 79, 77, 76, 78, 83,
             83, 84, 82, 78, 75, 72, 70, 69, 66, 65, 65, 63, 64, 66, 68,
             67, 69, 70, 75, 82, 88, 91, 92, 94, 99, 108, 114, 121, 124,
             126, 130, 135, 141, 144, 146, 151, 161, 170, 174, 172, 167,
             164, 168, 178, 184, 184, 186, 184, 188, 193, 192, 189, 191,
             197, 200, 202, 204, 208, 208, 207, 204, 202, 205, 202, 199,
             198, 197, 196, 196, 196, 196, 196, 194, 193, 191, 192, 194,
             193, 192, 190, 189, 190, 191, 190, 191, 189, 188, 189, 191,
             190, 190, 191, 191, 191, 189, 189, 190, 189, 188, 188, 189,
             189, 190, 188, 188, 187, 188, 188, 189, 189, 189, 187, 188,
             187, 187, 189, 189, 189, 189, 189, 188, 187, 187, 188, 189,
             189, 189, 189, 188, 188, 188, 188, 188, 189, 189, 188, 188,
             188, 188, 188, 188, 188, 188, 189, 188, 189, 189, 187, 187,
             188, 188, 188, 189, 188, 188, 187, 187, 187, 188, 189, 188,
             187, 188, 188, 189};
const uint16_t gestoZ[] PROGMEM = {127, 126, 126, 126, 124, 125, 125, 126, 127, 126, 126,
127,
             128, 127, 128, 125, 125, 124, 126, 125, 130, 130, 132, 136,
             138, 141, 143, 145, 145, 146, 148, 149, 148, 150, 150, 149,
             151, 151, 152, 154, 155, 155, 159, 160, 161, 161, 163, 164,
             165, 166, 168, 170, 169, 171, 168, 170, 172, 172, 172, 172,
             172, 170, 171, 176, 177, 181, 180, 180, 174, 174, 174, 172,
             174, 176, 179, 179, 177, 177, 176, 175, 177, 177, 177, 177,
             176, 180, 181, 182, 182, 182, 181, 181, 181, 182, 182, 182,
             181, 179, 179, 179, 179, 175, 171, 167, 166, 166, 167, 166,
             165, 158, 153, 153, 156, 157, 156, 152, 149, 148, 151, 147,
             150, 143, 140, 141, 142, 140, 137, 134, 131, 131, 131, 131,
             133, 134, 136, 137, 137, 135, 133, 132, 131, 131, 131, 130,
             129, 130, 130, 131, 131, 129, 127, 130, 129, 128, 128, 129,
             128, 129, 127, 127, 127, 127, 128, 127, 127, 127, 128, 127,
             126, 127, 127, 127, 128, 125, 127, 126, 127, 127, 127, 127,
             127, 127, 127, 128, 127, 127, 126, 126, 127, 126, 127, 127,
             127, 127, 127, 126, 126, 126, 127, 127, 126, 127, 127, 126,
             127, 127, 126, 126, 126, 126, 126, 127, 127, 127, 127, 127,
             127, 126, 126, 127, 127, 127, 127, 127, 126, 128, 128, 127,
             127, 127, 127, 127, 127, 127, 127, 127, 127, 126, 127};
```

```
int i = 0;

int n = 240;
int READY = 2;
int END = 3;
char rxChar;
double varX=0;
double varY=0;
double varZ=0;
double patronX=0;
double patronY=0;
double patronZ=0;
double Xxy=0;
double Yxy=0;
double Zxy=0;
double Xxsquare=0;
double Yxsquare=0;
double Zxsquare=0;
double Xysquare=0;
double Yysquare=0;
double Zysquare=0;
double Xxsum=0;
double Yxsum=0;
double Zxsum=0;
double Xysum=0;
double Yysum=0;
double Zysum = 0;
double Xxysum=0;
double Yxysum=0;
double Zxysum=0;
double Xxsqr_sum=0;
double Yxsqr_sum=0;
double Zxsqr_sum=0;
double Xysqr_sum=0;
double Yysqr_sum=0;
double Zysqr_sum=0;
double numX=0;
double denoX=0;
double numY=0;
double denoY=0;
double numZ=0;
double denoZ=0;
double coeffX=0;
double coeffY=0;
double coeffZ=0;


void setup() {
  // Inicializar comunicación por puerto serie a 115200 bits por segundo
  Serial.begin(115200);
  pinMode(A10, INPUT);
  pinMode(A9, INPUT);
  pinMode(A7, INPUT);
}

void loop() {

  if(Serial.available())
    {
    rxChar = Serial.read();
    if(rxChar=='1')
    {
      for (i=0; i<n; i++)
      {
        //Lectura del patrón
        patronX=pgm_read_word_near(patronIzqX + i);
        patronY=pgm_read_word_near(patronIzqY + i);
        patronZ=pgm_read_word_near(patronIzqZ + i);
```

```
        //Lectura de dato "leído" de prueba.
        varX=pgm_read_word_near(gestoX + i);
        varY=pgm_read_word_near(gestoY + i);
        varZ=pgm_read_word_near(gestoZ + i);

        //Correlacion cruzada eje X
        Xxy= varX * patronX;
        Xxsquare = varX * varX;
        Xysquare = patronX * patronX;
        Xxsum = Xxsum + varX;
        Xysum = Xysum + patronX;
        Xxysum = Xxysum + Xxy;
        Xxsqr_sum = Xxsqr_sum + Xxsquare;
        Xysqr_sum = Xysqr_sum + Xysquare;

        //Correlacion cruzada eje Y
        Yxy= varY * patronY;
        Yxsquare = varY * varY;
        Yysquare = patronY * patronY;
        Yxsum = Yxsum + varY;
        Yysum = Yysum + patronY;
        Yxysum = Yxysum + Yxy;
        Yxsqr_sum = Yxsqr_sum + Yxsquare;
        Yysqr_sum = Yysqr_sum + Yysquare;

        //Correlacion cruzada eje Z
        Zxy = varZ * patronZ;
        Zxsquare = varZ * varZ;
        Zysquare = patronZ * patronZ;
        Zxsum = Zxsum + varZ;
        Zysum = Zysum + patronZ;
        Zxysum = Zxysum + Zxy;
        Zxsqr_sum = Zxsqr_sum + Zxsquare;
        Zysqr_sum = Zysqr_sum + Zysquare;

      }

      //Coeficiente Correlación Cruzada X: GESTO GIRO DERECHA
      numX=1.0*((n*Xxysum)-(Xxsum*Xysum));
      denoX=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
      coeffX=numX/sqrt(denoX);

      //Coeficiente Correlacion Cruzada Y: GESTO GIRO DERECHA
      numY=1.0*((n*Yxysum)-(Yxsum*Yysum));
      denoY=1.0*((n*Yxsqr_sum-Yxsum*Yxsum)*(n*Yysqr_sum-Yysum*Yysum));
      coeffY=numY/sqrt(denoY);

      //Coeficiente Correlacion Cruzada Z: GESTO GIRO DERECHA
      numZ=1.0*((n*Zxysum)-(Zxsum*Zysum));
      denoZ=1.0*((n*Zxsqr_sum-Zxsum*Zxsum)*(n*Zysqr_sum-Zysum*Zysum));
      coeffZ=numZ/sqrt(denoZ);

      Serial.println(READY);
      Serial.println(coeffX);
      Serial.println(coeffY);
      Serial.println(coeffZ);
      Serial.println(END);
    }
  }
  delay(80);
}
```

A.3 Código 4. Probando el desplazamiento.

despl_cc_arduino.ino

```
#include <avr/pgmspace.h>
#include <stdint.h>
#include <math.h>

char rxChar;

//PATRÓN
const uint16_t GiroDerX[] PROGMEM = {85, 86, 85, 86, 85, 86, 86, 86, 83, 87, 87, 87, 85,
84, 84,
              87, 85, 85, 86, 86, 85, 85, 85, 85, 86, 85, 86, 85, 85, 86,
              85, 86, 85, 85, 86, 86, 86, 84, 86, 85, 84, 84, 81, 81, 77,
              72, 67, 64, 62, 66, 69, 70, 70, 72, 72, 72, 73, 76, 78, 80,
              82, 84, 86, 88, 89, 87, 87, 89, 91, 95, 96, 93, 92, 96, 98,
              99, 99, 101, 105, 103, 105, 107, 111, 111, 112, 113, 112,
              116, 119, 122, 123, 122, 124, 128, 126, 126, 128, 131, 131,
              131, 131, 132, 133, 134, 133, 134, 133, 135, 136, 136, 137,
              139, 138, 139, 140, 141, 140, 140, 140, 139, 138, 137, 137,
              136, 135, 136, 134, 133, 133, 133, 132, 132, 133, 132, 133,
              134, 134, 134, 135, 133, 133, 134, 133, 133, 132, 132, 133,
              134, 134, 133, 134, 134, 134, 134, 134, 133, 134, 134, 132,
              133, 132, 133, 132, 133, 133, 132, 134, 133, 134, 134, 134,
              134, 134, 135, 136, 135, 135, 134, 135, 136, 137, 138, 138,
              139, 139, 138, 138, 136, 135, 133, 131, 128, 127, 124, 124,
              121, 120, 112, 112, 111, 108, 107, 107, 107, 109, 110, 110,
              105, 101, 97, 95, 92, 90, 87, 82, 77, 77, 76, 73, 70, 70,
              70, 68, 67, 68, 69, 70, 74, 74, 72, 73, 74, 74, 78, 77, 79,
              79, 82, 84, 86};
const uint16_t GiroDerY[] PROGMEM = {144, 144, 145, 144, 144, 145, 144, 145, 147, 144, 145,
144,
              145, 145, 144, 143, 146, 146, 144, 145, 144, 144, 144, 144,
              145, 144, 144, 145, 144, 145, 144, 144, 144, 145, 144, 144,
              144, 144, 144, 144, 144, 144, 142, 142, 142, 144, 142, 146,
              150, 153, 152, 151, 156, 158, 155, 150, 149, 148, 148, 147,
              148, 148, 146, 148, 144, 147, 148, 147, 147, 147, 146, 147,
              147, 141, 141, 140, 138, 136, 135, 134, 130, 133, 131, 129,
              128, 125, 125, 122, 122, 120, 117, 117, 118, 117, 116, 115,
              113, 113, 113, 112, 113, 114, 115, 115, 115, 116, 116, 117,
              118, 118, 118, 117, 115, 117, 118, 119, 119, 118, 119, 119,
              120, 121, 122, 121, 122, 121, 120, 121, 120, 120, 120, 120,
              119, 119, 118, 118, 119, 119, 118, 118, 119, 120, 120, 119,
              119, 118, 119, 119, 119, 119, 120, 119, 118, 119, 119, 120,
              119, 119, 119, 119, 119, 119, 119, 119, 119, 120, 119, 119,
              119, 119, 118, 117, 118, 117, 117, 117, 118, 118, 118, 118,
              118, 118, 120, 119, 121, 121, 120, 120, 122, 121, 121, 121,
              122, 124, 123, 121, 120, 119, 123, 124, 127, 127, 128, 128,
              129, 133, 136, 138, 140, 137, 135, 136, 137, 139, 141, 142,
              143, 142, 145, 149, 148, 144, 143, 138, 140, 140, 140, 140,
              139, 138, 141, 140, 141, 140, 140, 137, 134, 135, 137, 140};
const uint16_t GiroDerZ[] PROGMEM = {124, 124, 124, 125, 125, 124, 125, 123, 124, 126, 125,
125,
              125, 125, 125, 125, 124, 125, 124, 125, 124, 125, 125, 125,
              125, 125, 125, 125, 124, 124, 125, 125, 126, 125, 126, 125,
              126, 126, 125, 125, 127, 127, 127, 127, 129, 129, 129, 129,
              127, 128, 128, 127, 127, 128, 130, 130, 131, 132, 136, 137,
              139, 140, 142, 141, 147, 146, 149, 153, 153, 154, 156, 157,
              156, 158, 159, 161, 163, 164, 165, 165, 169, 168, 169, 170,
              170, 172, 172, 174, 175, 178, 179, 179, 180, 183, 183, 184,
              186, 187, 188, 190, 191, 191, 191, 191, 192, 191, 191, 191,
              192, 193, 193, 193, 193, 193, 194, 194, 193, 193, 192, 193,
              194, 193, 193, 193, 194, 192, 193, 193, 194, 193, 194, 193,
              193, 194, 193, 193, 193, 193, 192, 192, 192, 192, 192, 192,
              191, 193, 192, 192, 191, 192, 192, 192, 192, 192, 192, 192,
              192, 193, 192, 191, 193, 192, 192, 192, 193, 192, 193, 193,
              192, 192, 191, 191, 191, 192, 192, 191, 192, 192, 192, 193,
              192, 193, 192, 191, 190, 189, 189, 186, 187, 184, 183, 183,
              183, 180, 180, 180, 176, 176, 175, 174, 173, 173, 172, 170,
```

```
              168, 166, 164, 161, 156, 154, 153, 148, 146, 142, 139, 137,
              135, 132, 128, 127, 128, 127, 128, 127, 128, 129, 129, 130,
              131, 129, 129, 129, 129, 128, 130, 131, 129, 130, 129, 128};

//GESTO_PRUEBA
const uint16_t gestoX[] PROGMEM = {86, 86, 87, 86, 88, 88, 88, 87, 87, 87, 86, 87, 86, 86,
87,
             87, 87, 87, 87, 87, 87, 88, 87, 87, 87, 88, 87, 87, 87, 87,
             88, 87, 86, 85, 86, 85, 83, 80, 76, 75, 72, 72, 71, 74, 74,
             72, 71, 71, 71, 74, 76, 80, 85, 87, 90, 93, 96, 97, 102, 105,
             106, 109, 109, 110, 112, 115, 116, 114, 115, 115, 115, 117,
             120, 123, 123, 125, 126, 128, 131, 132, 132, 134, 135, 138,
             139, 141, 139, 138, 140, 141, 142, 141, 141, 141, 141, 141,
             140, 139, 138, 139, 138, 138, 139, 139, 139, 139, 139, 139,
             139, 140, 139, 138, 138, 138, 137, 136, 136, 136, 135, 135,
             135, 134, 136, 136, 137, 135, 137, 137, 137, 137, 137, 137,
             136, 136, 137, 137, 137, 137, 137, 137, 138, 139, 138, 138,
             138, 137, 137, 137, 137, 136, 138, 137, 138, 137, 138, 137,
             139, 138, 139, 138, 139, 139, 139, 140, 140, 142, 142, 141,
             143, 143, 143, 143, 143, 141, 142, 139, 137, 136, 133, 132,
             131, 128, 129, 129, 129, 126, 124, 122, 120, 121, 120, 117,
             118, 114, 107, 105, 103, 104, 105, 103, 98, 95, 90, 85, 80,
             79, 76, 75, 75, 69, 66, 60, 61, 62, 65, 70, 75, 78, 80, 82,
             85, 87, 88, 89, 88, 87, 85, 85, 83, 81, 82, 83, 86, 87, 86,
             87, 86, 85, 84, 83};
const uint16_t gestoY[] PROGMEM = {147, 148, 149, 149, 149, 149, 149, 148, 148, 148, 148,
148,
             148, 148, 149, 148, 148, 148, 149, 148, 148, 148, 148, 148,
             147, 147, 148, 148, 148, 149, 148, 148, 148, 147, 147, 146,
             146, 145, 144, 143, 145, 146, 149, 153, 157, 160, 161, 161,
             159, 160, 162, 156, 157, 159, 158, 154, 151, 148, 145, 146,
             145, 144, 142, 139, 139, 136, 136, 138, 137, 136, 135, 134,
             132, 130, 131, 128, 127, 127, 125, 122, 121, 119, 117, 114,
             115, 119, 123, 127, 126, 125, 125, 124, 125, 127, 127, 126,
             123, 123, 125, 124, 123, 124, 123, 123, 124, 123, 123, 122,
             122, 124, 124, 124, 124, 124, 123, 123, 122, 122, 121, 121,
             122, 122, 122, 123, 121, 123, 123, 123, 123, 122, 122, 122,
             123, 122, 122, 123, 121, 122, 123, 123, 123, 123, 122, 123,
             122, 121, 122, 123, 122, 123, 122, 122, 122, 121, 122, 122,
             123, 122, 123, 123, 124, 123, 122, 122, 123, 123, 123, 124,
             123, 123, 124, 123, 121, 120, 120, 120, 122, 122, 124, 126,
             125, 127, 128, 127, 129, 132, 134, 135, 134, 133, 132, 128,
             134, 136, 137, 136, 133, 135, 134, 133, 136, 139, 144, 146,
             147, 141, 138, 138, 141, 144, 147, 154, 152, 144, 137, 135,
             137, 141, 143, 144, 143, 140, 137, 139, 145, 148, 146, 146,
             146, 147, 150, 148, 145, 143, 145, 147, 146, 144, 143, 142};
const uint16_t gestoZ[] PROGMEM = {140, 139, 141, 140, 140, 140, 140, 140, 140, 140, 140,
140,
             140, 140, 140, 140, 140, 141, 140, 142, 141, 140, 140, 139,
             139, 141, 139, 140, 140, 141, 140, 140, 140, 140, 140, 141,
             140, 140, 140, 142, 142, 141, 141, 140, 136, 138, 141, 146,
             147, 147, 147, 147, 150, 151, 152, 155, 157, 158, 160, 161,
             162, 165, 166, 166, 167, 169, 170, 170, 172, 173, 173, 173,
             175, 179, 178, 180, 181, 182, 184, 185, 187, 187, 187, 189,
             192, 197, 197, 196, 196, 198, 198, 198, 198, 199, 198, 196,
             196, 195, 195, 194, 194, 194, 194, 194, 193, 193, 193, 194,
             194, 194, 193, 193, 193, 193, 193, 194, 193, 193, 193, 193,
             194, 193, 193, 194, 193, 194, 193, 193, 193, 193, 193, 193,
             193, 193, 193, 193, 193, 195, 194, 194, 194, 194, 194, 193,
             193, 194, 194, 193, 193, 194, 193, 193, 194, 194, 193, 194,
             194, 194, 194, 193, 193, 193, 193, 193, 192, 194, 194, 193,
             193, 192, 193, 192, 193, 193, 193, 193, 192, 193, 192, 189,
             190, 188, 188, 186, 186, 183, 181, 179, 179, 178, 176, 178,
             174, 170, 169, 167, 167, 166, 166, 163, 159, 157, 153, 150,
             150, 149, 149, 146, 144, 141, 142, 138, 138, 138, 140, 139,
             137, 137, 137, 136, 138, 137, 138, 138, 137, 138, 138, 139,
             136, 137, 137, 137, 137, 137, 137, 136, 136, 136, 137, 136};

int i = 0;
```

```
int j=0;

int n = 240;
int cuenta = 2;
int READY = 2;
int END = 3;

double varX=0;
double varY=0;
double varZ=0;
uint16_t patronX=0;
uint16_t patronY=0;
uint16_t patronZ=0;
double Xxy=0;
double Yxy=0;
double Zxy=0;
double Xxsquare=0;
double Yxsquare=0;
double Zxsquare=0;
double Xysquare=0;
double Yysquare=0;
double Zysquare=0;
double Xxsum=0;
double Yxsum=0;
double Zxsum=0;
double Xysum=0;
double Yysum=0;
double Zysum = 0;
double Xxysum=0;
double Yxysum=0;
double Zxysum=0;
double Xxsqr_sum=0;
double Yxsqr_sum=0;
double Zxsqr_sum=0;
double Xysqr_sum=0;
double Yysqr_sum=0;
double Zysqr_sum=0;
double numX=0;
double denoX=0;
double numY=0;
double denoY=0;
double numZ=0;
double denoZ=0;
double coeffX=0;
double coeffY=0;
double coeffZ=0;

uint16_t auxX [240];
uint16_t auxY [240];
uint16_t auxZ [240];


void setup() {
  // Inicializar comunicación por puerto serie a 115200 bits por segundo
  Serial.begin(115200);
  pinMode(A10, INPUT);
  pinMode(A9, INPUT);
  pinMode(A7, INPUT);

}


void loop() {

  if(Serial.available())
  {
    rxChar = Serial.read();
    if(rxChar=='1')
    {
      if (j==0)
```

```
    {
      //PRIMERO LLENAMOS LOS VECTORES DE DATOS QUE "RECIBO"
      for(i=0;i<n;i++)
      {
        auxX[i]=pgm_read_word_near(gestoX + i);
        auxY[i]=pgm_read_word_near(gestoY + i);
        auxZ[i]=pgm_read_word_near(gestoZ + i);
      }
      j=1;
    }

    //COMPARACIÓN ENTRE EL VECTOR DE DATOS Y EL PATRÓN CADA 2 MUESTRAS
    if (cuenta==2)
    {
      Xxsum=0;
      Xysum=0;
      Xxysum=0;
      Xxsqr_sum=0;
      Xysqr_sum=0;

      Yxsum=0;
      Yysum=0;
      Yxysum=0;
      Yxsqr_sum=0;
      Yysqr_sum=0;

      Zxsum=0;
      Zysum=0;
      Zxysum=0;
      Zxsqr_sum=0;
      Zysqr_sum=0;

      for (i=0; i<n; i++)
      {
        //Lectura del patrón
        patronX=pgm_read_word_near(GiroDerX + i);
        patronY=pgm_read_word_near(GiroDerY + i);
        patronZ=pgm_read_word_near(GiroDerZ + i);

        //Correlacion cruzada eje X
        Xxy= auxX[i] * patronX;
        Xxsquare = auxX[i] * auxX[i];
        Xysquare = patronX * patronX;
        Xxsum = Xxsum + auxX[i];
        Xysum = Xysum + patronX;
        Xxysum = Xxysum + Xxy;
        Xxsqr_sum = Xxsqr_sum + Xxsquare;
        Xysqr_sum = Xysqr_sum + Xysquare;

        //Correlacion cruzada eje Y
        Yxy= auxY[i] * patronY;
        Yxsquare = auxY[i] * auxY[i];
        Yysquare = patronY * patronY;
        Yxsum = Yxsum + auxY[i];
        Yysum = Yysum + patronY;
        Yxysum = Yxysum + Yxy;
        Yxsqr_sum = Yxsqr_sum + Yxsquare;
        Yysqr_sum = Yysqr_sum + Yysquare;

        //Correlacion cruzada eje Z
        Zxy = auxZ[i] * patronZ;
        Zxsquare = auxZ[i] * auxZ[i];
        Zysquare = patronZ * patronZ;
        Zxsum = Zxsum + auxZ[i];
        Zysum = Zysum + patronZ;
        Zxysum = Zxysum + Zxy;
        Zxsqr_sum = Zxsqr_sum + Zxsquare;
        Zysqr_sum = Zysqr_sum + Zysquare;
      }
```

```
        //Coeficiente Correlación Cruzada X: GESTO GIRO DERECHA
        numX=1.0*((n*Xxysum)-(Xxsum*Xysum));
        denoX=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
        coeffX=numX/sqrt(denoX);

        //Coeficiente Correlacion Cruzada Y: GESTO GIRO DERECHA
        numY=1.0*((n*Yxysum)-(Yxsum*Yysum));
        denoY=1.0*((n*Yxsqr_sum-Yxsum*Yxsum)*(n*Yysqr_sum-Yysum*Yysum));
        coeffY=numY/sqrt(denoY);

        //Coeficiente Correlacion Cruzada Z: GESTO GIRO DERECHA
        numZ=((n*Zxysum)-(Zxsum*Zysum));
        denoZ=((n*Zxsqr_sum-Zxsum*Zxsum)*(n*Zysqr_sum-Zysum*Zysum));
        coeffZ=numZ/sqrt(denoZ);

        Serial.println(READY);
        Serial.println(coeffX);
        Serial.println(coeffY);
        Serial.println(coeffZ);

        for(i=0;i<10;i++)
        {
          Serial.println(auxX[n-(10-i)]);
        }
        for(i=0;i<10;i++)
        {
          Serial.println(auxY[n-(10-i)]);
        }
        for(i=0;i<10;i++)
        {
          Serial.println(auxZ[n-(10-i)]);
        }

        Serial.println(END);
        cuenta = 0;
      }else
      {
        cuenta++;
        for(i=0;i<n;i++)
        {
          if(i==(n-1))
          {
            auxX[i]=0;
            auxY[i]=0;
            auxZ[i]=0;
          }else
          {
            auxX[i]=auxX[i+1];
            auxY[i]=auxY[i+1];
            auxZ[i]=auxZ[i+1];
          }
        }
      }
    }
  }
  delay(16);
}
```

A.4 Código 5. Captura de gesto, correlación en FLORA y envío de datos a MATLAB®

lectura_cc_arduino.ino

```
#include <avr/pgmspace.h>
#include <stdint.h>
#include <math.h>

int ledpin = 7;
unsigned long time;
char rxChar;

//PATRÓN
const uint16_t GiroDerX[] PROGMEM = {85, 86, 85, 86, 85, 86, 86, 86, 83, 87, 87, 87, 85,
84, 84,
              87, 85, 85, 86, 86, 85, 85, 85, 85, 86, 85, 86, 85, 85, 86,
              85, 86, 85, 85, 86, 86, 86, 84, 86, 85, 84, 84, 81, 81, 77,
              72, 67, 64, 62, 66, 69, 70, 70, 72, 72, 72, 73, 76, 78, 80,
              82, 84, 86, 88, 89, 87, 87, 89, 91, 95, 96, 93, 92, 96, 98,
              99, 99, 101, 105, 103, 105, 107, 111, 111, 112, 113, 112,
              116, 119, 122, 123, 122, 124, 128, 126, 126, 128, 131, 131,
              131, 131, 132, 133, 134, 133, 134, 133, 135, 136, 136, 137,
              139, 138, 139, 140, 141, 140, 140, 140, 139, 138, 137, 137,
              136, 135, 136, 134, 133, 133, 133, 132, 132, 133, 132, 133,
              134, 134, 134, 135, 133, 133, 134, 133, 133, 132, 132, 133,
              134, 134, 133, 134, 134, 134, 134, 134, 133, 134, 134, 132,
              133, 132, 133, 132, 133, 133, 132, 134, 133, 134, 134, 134,
              134, 134, 135, 136, 135, 135, 134, 135, 136, 137, 138, 138,
              139, 139, 138, 138, 136, 135, 133, 131, 128, 127, 124, 124,
              121, 120, 112, 112, 111, 108, 107, 107, 107, 109, 110, 110,
              105, 101, 97, 95, 92, 90, 87, 82, 77, 77, 76, 73, 70, 70,
              70, 68, 67, 68, 69, 70, 74, 74, 72, 73, 74, 74, 78, 77, 79,
              79, 82, 84, 86};
const uint16_t GiroDerY[] PROGMEM = {144, 144, 145, 144, 144, 145, 144, 145, 147, 144, 145,
144,
              145, 145, 144, 143, 146, 146, 144, 145, 144, 144, 144, 144,
              145, 144, 144, 145, 144, 145, 144, 144, 144, 145, 144, 144,
              144, 144, 144, 144, 144, 144, 142, 142, 142, 144, 142, 146,
              150, 153, 152, 151, 156, 158, 155, 150, 149, 148, 148, 147,
              148, 148, 146, 148, 144, 147, 148, 147, 147, 147, 146, 147,
              147, 141, 141, 140, 138, 136, 135, 134, 130, 133, 131, 129,
              128, 125, 125, 122, 122, 120, 117, 117, 118, 117, 116, 115,
              113, 113, 113, 112, 113, 114, 115, 115, 115, 116, 116, 117,
              118, 118, 118, 117, 115, 117, 118, 119, 119, 118, 119, 119,
              120, 121, 122, 121, 122, 121, 120, 121, 120, 120, 120, 120,
              119, 119, 118, 118, 119, 119, 118, 118, 119, 120, 120, 119,
              119, 118, 119, 119, 119, 119, 120, 119, 118, 119, 119, 120,
              119, 119, 119, 119, 119, 119, 119, 119, 119, 120, 119, 119,
              119, 119, 118, 117, 118, 117, 117, 117, 118, 118, 118, 118,
              118, 118, 120, 119, 121, 121, 120, 120, 122, 121, 121, 121,
              122, 124, 123, 121, 120, 119, 123, 124, 127, 127, 128, 128,
              129, 133, 136, 138, 140, 137, 135, 136, 137, 139, 141, 142,
              143, 142, 145, 149, 148, 144, 143, 138, 140, 140, 140, 140,
              139, 138, 141, 140, 141, 140, 140, 137, 134, 135, 137, 140};
const uint16_t GiroDerZ[] PROGMEM = {124, 124, 124, 125, 125, 124, 125, 123, 124, 126, 125,
125,
              125, 125, 125, 125, 124, 125, 124, 125, 124, 125, 125, 125,
              125, 125, 125, 125, 124, 124, 125, 125, 126, 125, 126, 125,
              126, 126, 125, 125, 127, 127, 127, 127, 129, 129, 129, 129,
              127, 128, 128, 127, 127, 128, 130, 130, 131, 132, 136, 137,
              139, 140, 142, 141, 147, 146, 149, 153, 153, 154, 156, 157,
              156, 158, 159, 161, 163, 164, 165, 165, 169, 168, 169, 170,
              170, 172, 172, 174, 175, 178, 179, 179, 180, 183, 183, 184,
              186, 187, 188, 190, 191, 191, 191, 191, 192, 191, 191, 191,
              192, 193, 193, 193, 193, 193, 194, 194, 193, 193, 192, 193,
              194, 193, 193, 193, 194, 192, 193, 193, 194, 193, 194, 193,
              193, 194, 193, 193, 193, 193, 192, 192, 192, 192, 192, 192,
              191, 193, 192, 192, 191, 192, 192, 192, 192, 192, 192, 192,
```

```
              192, 193, 192, 191, 193, 192, 192, 192, 193, 192, 193, 193,
              192, 192, 191, 191, 191, 192, 192, 191, 192, 192, 192, 193,
              192, 193, 192, 191, 190, 189, 189, 186, 187, 184, 183, 183,
              183, 180, 180, 180, 176, 176, 175, 174, 173, 173, 172, 170,
              168, 166, 164, 161, 156, 154, 153, 148, 146, 142, 139, 137,
              135, 132, 128, 127, 128, 127, 128, 127, 128, 129, 129, 130,
              131, 129, 129, 129, 129, 128, 130, 131, 129, 130, 129, 128};


int i = 0;
int j=0;
int n = 240;
int cuenta = 20;
int READY = 2;
int END = 3;

double varX=0;
double varY=0;
double varZ=0;
uint16_t patronX=0;
uint16_t patronY=0;
uint16_t patronZ=0;
double Xxy=0;
double Yxy=0;
double Zxy=0;
double Xxsquare=0;
double Yxsquare=0;
double Zxsquare=0;
double Xysquare=0;
double Yysquare=0;
double Zysquare=0;
double Xxsum=0;
double Yxsum=0;
double Zxsum=0;
double Xysum=0;
double Yysum=0;
double Zysum = 0;
double Xxysum=0;
double Yxysum=0;
double Zxysum=0;
double Xxsqr_sum=0;
double Yxsqr_sum=0;
double Zxsqr_sum=0;
double Xysqr_sum=0;
double Yysqr_sum=0;
double Zysqr_sum=0;
double numX=0;
double denoX=0;
double numY=0;
double denoY=0;
double numZ=0;
double denoZ=0;
double coeffX=0;
double coeffY=0;
double coeffZ=0;
uint16_t auxX [240];
uint16_t auxY [240];
uint16_t auxZ [240];

void setup() {
  // Inicializar comunicación por puerto serie a 115200 bits por segundo
  Serial.begin(115200);
  pinMode(A10, INPUT);
  pinMode(A9, INPUT);
  pinMode(A7, INPUT);

}
```

```
void loop() {
  if(Serial.available())
  {
    rxChar = Serial.read();
    if(rxChar=='1')
    {
      if (j==0)
      {
        //PRIMERO LLENAMOS LOS VECTORES DE DATOS QUE "RECIBO"
        for(i=0;i<n;i++)
        {
          auxX[i]=analogRead(A10)/4;
          auxY[i]=analogRead(A9)/4;
          auxZ[i]=analogRead(A7)/4;
        }
        j=1;
      }

      //COMPARACIÓN ENTRE EL VECTOR DE DATOS Y EL PATRÓN CADA 20 MUESTRAS
      if (cuenta==20)
      {
        Xxsum=0;
        Xysum=0;
        Xxysum=0;
        Xxsqr_sum=0;
        Xysqr_sum=0;

        Yxsum=0;
        Yysum=0;
        Yxysum=0;
        Yxsqr_sum=0;
        Yysqr_sum=0;

        Zxsum=0;
        Zysum=0;
        Zxysum=0;
        Zxsqr_sum=0;
        Zysqr_sum=0;

        for (i=0; i<n; i++)
        {
          //Lectura del patrón
          patronX=pgm_read_word_near(GiroDerX + i);
          patronY=pgm_read_word_near(GiroDerY + i);
          patronZ=pgm_read_word_near(GiroDerZ + i);

          //Correlacion cruzada eje X
          Xxy= auxX[i] * patronX;
          Xxsquare = auxX[i] * auxX[i];
          Xysquare = patronX * patronX;
          Xxsum = Xxsum + auxX[i];
          Xysum = Xysum + patronX;
          Xxysum = Xxysum + Xxy;
          Xxsqr_sum = Xxsqr_sum + Xxsquare;
          Xysqr_sum = Xysqr_sum + Xysquare;

          //Correlacion cruzada eje Y
          Yxy= auxY[i] * patronY;
          Yxsquare = auxY[i] * auxY[i];
          Yysquare = patronY * patronY;
          Yxsum = Yxsum + auxY[i];
          Yysum = Yysum + patronY;
          Yxysum = Yxysum + Yxy;
          Yxsqr_sum = Yxsqr_sum + Yxsquare;
          Yysqr_sum = Yysqr_sum + Yysquare;
```

```
          //Correlacion cruzada eje Z
          Zxy = auxZ[i] * patronZ;
          Zxsquare = auxZ[i] * auxZ[i];
          Zysquare = patronZ * patronZ;
          Zxsum = Zxsum + auxZ[i];
          Zysum = Zysum + patronZ;
          Zxysum = Zxysum + Zxy;
          Zxsqr_sum = Zxsqr_sum + Zxsquare;
          Zysqr_sum = Zysqr_sum + Zysquare;
        }
        //Coeficiente Correlación Cruzada X: GESTO GIRO DERECHA
        numX=1.0*((n*Xxysum)-(Xxsum*Xysum));
        denoX=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
        coeffX=numX/sqrt(denoX);

        //Coeficiente Correlacion Cruzada Y: GESTO GIRO DERECHA
        numY=1.0*((n*Yxysum)-(Yxsum*Yysum));
        denoY=1.0*((n*Yxsqr_sum-Yxsum*Yxsum)*(n*Yysqr_sum-Yysum*Yysum));
        coeffY=numY/sqrt(denoY);

        //Coeficiente Correlacion Cruzada Z: GESTO GIRO DERECHA
        numZ=((n*Zxysum)-(Zxsum*Zysum));
        denoZ=((n*Zxsqr_sum-Zxsum*Zxsum)*(n*Zysqr_sum-Zysum*Zysum));
        coeffZ=numZ/sqrt(denoZ);

        if (coeffX>0.8 && coeffY>0.8 && coeffZ>0.8)
        {
          digitalWrite(ledpin, HIGH);
          Serial.println(READY);
          Serial.println(coeffX);
          Serial.println(coeffY);
          Serial.println(coeffZ);
          Serial.println(END);

          delay(2500);
          digitalWrite(ledpin,LOW);
          }else
          {
            digitalWrite(ledpin, LOW);
            Serial.println(READY);
            Serial.println(coeffX);
            Serial.println(coeffY);
            Serial.println(coeffZ);
            Serial.println(END);
          }
          cuenta = 0;
        }else
        {
          cuenta++;
          for(i=0;i<n;i++)
          {
            if(i==(n-1))
            {
              auxX[i]=analogRead(A10)/4;
              auxY[i]=analogRead(A9)/4;
              auxZ[i]=analogRead(A7)/4;
            }else
            {
              auxX[i]=auxX[i+1];
              auxY[i]=auxY[i+1];
              auxZ[i]=auxZ[i+1];
            }
          }
        }
      }
    }
  delay(35);
}
```

A.5 Código 6. Captura de gesto, correlación en FLORA y encendido de LED.

lectura_cc_arduino_led.ino

```
#include <avr/pgmspace.h>
#include <stdint.h>
#include <math.h>

int ledpin = 7;
unsigned long time;
char rxChar;

//PATRÓN DERECHO
const uint16_t GiroDerX[] PROGMEM = {83, 84, 84, 83, 84, 83, 84, 83, 84, 83, 84, 83, 83,
83,
                83, 83, 83, 84, 84, 84, 83, 84, 84, 84, 84, 83, 82, 83,
                84, 82, 81, 80, 78, 76, 74, 72, 66, 66, 66, 62, 60, 58,
                57, 59, 63, 66, 70, 75, 76, 78, 83, 85, 91, 92, 95, 96,
                99, 104, 109, 109, 114, 116, 120, 124, 125, 132, 133, 136,
                134, 141, 142, 143, 147, 150, 151, 151, 153, 149, 157,
                154, 157, 155, 153, 150, 148, 147, 146, 142, 140, 138,
                140, 139, 138, 138, 137, 137, 139, 138, 140, 140, 142,
                141, 140, 140, 141, 142, 143, 143, 142, 141, 140, 142,
                141, 140, 140, 140, 140, 139, 139, 140, 140, 140, 141,
                141, 142, 144, 147, 149, 152, 150, 148, 149, 148, 148,
                147, 147, 148, 145, 142, 141, 138, 137, 136, 134, 137,
                133, 131, 129, 119, 109, 102, 102, 108, 110, 106, 100,
                97, 88, 84, 80, 78, 76, 77, 73, 69, 64, 67, 66, 70, 68,
                68, 64, 65, 68, 67, 52, 59, 79, 86, 88, 88, 84, 83, 85,
                83, 82, 82, 81, 82, 84, 84, 84, 84, 84, 83, 83, 82, 82,
                84, 84, 83, 83, 83, 82, 83, 83, 83, 82, 83, 84, 83, 84,
                85, 84, 84, 82, 82, 83, 84, 83, 84, 83, 84, 83, 82, 82,
                82, 83, 83, 84, 84, 84, 83, 84, 84, 84, 84, 83, 83, 84};
const uint16_t GiroDerY[] PROGMEM = {136, 136, 136, 137, 136, 136, 136, 137, 136, 137, 137,
                136, 136, 136, 136, 137, 136, 138, 137, 137, 136, 136,
                136, 135, 137, 137, 136, 137, 136, 135, 136, 132, 133,
                134, 134, 136, 137, 141, 133, 129, 129, 131, 135, 136,
                137, 138, 140, 140, 141, 142, 143, 143, 142, 139, 139,
                138, 137, 136, 134, 133, 133, 133, 131, 127, 125, 124,
                127, 124, 120, 119, 117, 115, 112, 112, 114, 112, 109,
                109, 108, 108, 111, 112, 112, 110, 110, 112, 114, 114,
                116, 115, 115, 115, 117, 115, 114, 115, 115, 115, 115,
                115, 114, 114, 114, 114, 114, 114, 114, 115, 116, 114,
                114, 114, 114, 114, 114, 116, 116, 115, 116, 115, 114,
                114, 114, 114, 114, 113, 113, 111, 110, 108, 107, 110,
                112, 114, 113, 113, 115, 117, 118, 121, 122, 124, 125,
                125, 125, 128, 132, 133, 130, 129, 133, 137, 138, 139,
                138, 137, 137, 137, 139, 139, 144, 145, 144, 139, 141,
                140, 138, 135, 136, 137, 139, 137, 135, 133, 129, 133,
                142, 138, 135, 136, 133, 136, 137, 138, 138, 140, 139,
                139, 139, 137, 135, 136, 135, 135, 135, 136, 137, 137,
                135, 136, 135, 136, 136, 136, 135, 134, 135, 135, 136,
                137, 138, 138, 138, 137, 135, 136, 135, 135, 137, 136,
                137, 137, 137, 136, 136, 135, 136, 136, 136, 137, 137,
                137, 137, 138, 137, 137, 136, 135, 135, 136};
const uint16_t GiroDerZ[] PROGMEM = {130, 129, 129, 130, 129, 130, 130, 129, 129, 130, 129,
                130, 129, 129, 129, 130, 129, 129, 129, 129, 130, 129,
                130, 130, 130, 130, 129, 130, 131, 130, 130, 130, 131,
                129, 127, 128, 124, 122, 128, 130, 125, 125, 129, 133,
                129, 127, 132, 135, 133, 134, 135, 137, 139, 142, 144,
                146, 149, 154, 158, 161, 164, 167, 171, 173, 174, 179,
                182, 183, 183, 186, 188, 188, 191, 192, 193, 193, 197,
                194, 196, 194, 196, 196, 193, 194, 194, 193, 193, 192,
                192, 193, 192, 192, 192, 192, 192, 191, 193, 192, 193,
                192, 193, 192, 192, 193, 193, 193, 193, 193, 193, 193,
                193, 193, 193, 192, 193, 192, 193, 193, 192, 192, 192,
                193, 193, 193, 194, 194, 194, 195, 195, 197, 195, 193,
                193, 194, 192, 193, 191, 190, 189, 189, 186, 185, 184,
```

```
                184, 181, 179, 175, 171, 169, 163, 163, 164, 167, 162,
                151, 146, 147, 147, 142, 138, 134, 132, 131, 131, 128,
                124, 123, 129, 131, 126, 127, 129, 130, 129, 128, 122,
                125, 130, 130, 128, 130, 130, 130, 129, 129, 130, 130,
                129, 130, 131, 130, 129, 130, 130, 130, 130, 129, 129,
                129, 129, 130, 130, 130, 130, 130, 129, 130, 130, 130,
                129, 130, 129, 130, 130, 130, 130, 129, 129, 129, 129,
                130, 129, 129, 129, 129, 130, 130, 130, 130, 129, 130,
                129, 129, 130, 129, 129, 129, 129, 129, 129};

int i = 0;
int j=0;

int n = 240;
int cuenta = 50;

double varX=0;
double varY=0;
double varZ=0;
uint16_t patronX=0;
uint16_t patronY=0;
uint16_t patronZ=0;
double Xxy=0;
double Yxy=0;
double Zxy=0;
double Xxsquare=0;
double Yxsquare=0;
double Zxsquare=0;
double Xysquare=0;
double Yysquare=0;
double Zysquare=0;
double Xxsum=0;
double Yxsum=0;
double Zxsum=0;
double Xysum=0;
double Yysum=0;
double Zysum = 0;
double Xxysum=0;
double Yxysum=0;
double Zxysum=0;
double Xxsqr_sum=0;
double Yxsqr_sum=0;
double Zxsqr_sum=0;
double Xysqr_sum=0;
double Yysqr_sum=0;
double Zysqr_sum=0;
double numX=0;
double denoX=0;
double numY=0;
double denoY=0;
double numZ=0;
double denoZ=0;
double coeffX=0;
double coeffY=0;
double coeffZ=0;

uint16_t auxX [240];
uint16_t auxY [240];
uint16_t auxZ [240];


void setup() {
  pinMode(A10, INPUT);
  pinMode(A9, INPUT);
  pinMode(A7, INPUT);

}
```

```
void loop()
{
  if (j==0)
  {
    //PRIMERO LLENAMOS LOS VECTORES DE DATOS QUE "RECIBO"
    for(i=0;i<n;i++)
    {
      auxX[i]=analogRead(A10)/4;
      auxY[i]=analogRead(A9)/4;
      auxZ[i]=analogRead(A7)/4;
    }
    j=1;
  }
  //COMPARACIÓN ENTRE EL VECTOR DE DATOS Y EL PATRÓN CADA 10 MUESTRAS
  if (cuenta==50)
  {
    digitalWrite(ledpin, HIGH);
    delay(1);
    digitalWrite(ledpin, LOW);

    Xxsum=0;
    Xysum=0;
    Xxysum=0;
    Xxsqr_sum=0;
    Xysqr_sum=0;

    Yxsum=0;
    Yysum=0;
    Yxysum=0;
    Yxsqr_sum=0;
    Yysqr_sum=0;

    Zxsum=0;
    Zysum=0;
    Zxysum=0;
    Zxsqr_sum=0;
    Zysqr_sum=0;

    for (i=0; i<n; i++)
    {
      //Lectura del patrón
      patronX=pgm_read_word_near(GiroDerX + i);
      patronY=pgm_read_word_near(GiroDerY + i);
      patronZ=pgm_read_word_near(GiroDerZ + i);

      //Correlacion cruzada eje X
      Xxy= auxX[i] * patronX;
      Xxsquare = auxX[i] * auxX[i];
      Xysquare = patronX * patronX;
      Xxsum = Xxsum + auxX[i];
      Xysum = Xysum + patronX;
      Xxysum = Xxysum + Xxy;
      Xxsqr_sum = Xxsqr_sum + Xxsquare;
      Xysqr_sum = Xysqr_sum + Xysquare;

      //Correlacion cruzada eje Y
      Yxy= auxY[i] * patronY;
      Yxsquare = auxY[i] * auxY[i];
      Yysquare = patronY * patronY;
      Yxsum = Yxsum + auxY[i];
      Yysum = Yysum + patronY;
      Yxysum = Yxysum + Yxy;
      Yxsqr_sum = Yxsqr_sum + Yxsquare;
      Yysqr_sum = Yysqr_sum + Yysquare;

      //Correlacion cruzada eje Z
      Zxy = auxZ[i] * patronZ;
      Zxsquare = auxZ[i] * auxZ[i];
```

```
      Zysquare = patronZ * patronZ;
      Zxsum = Zxsum + auxZ[i];
      Zysum = Zysum + patronZ;
      Zxysum = Zxysum + Zxy;
      Zxsqr_sum = Zxsqr_sum + Zxsquare;
      Zysqr_sum = Zysqr_sum + Zysquare;
    }

  //Coeficiente Correlación Cruzada X: GESTO GIRO DERECHA
  numX=1.0*((n*Xxysum)-(Xxsum*Xysum));
  denoX=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
  coeffX=numX/sqrt(denoX);

  //Coeficiente Correlacion Cruzada Y: GESTO GIRO DERECHA
  numY=1.0*((n*Yxysum)-(Yxsum*Yysum));
  denoY=1.0*((n*Yxsqr_sum-Yxsum*Yxsum)*(n*Yysqr_sum-Yysum*Yysum));
  coeffY=numY/sqrt(denoY);

  //Coeficiente Correlacion Cruzada Z: GESTO GIRO DERECHA
  numZ=((n*Zxysum)-(Zxsum*Zysum));
  denoZ=((n*Zxsqr_sum-Zxsum*Zxsum)*(n*Zysqr_sum-Zysum*Zysum));
  coeffZ=numZ/sqrt(denoZ);

  if (coeffX>0.7 && coeffY>0.7 && coeffZ>0.7)
  {
    digitalWrite(ledpin, HIGH);
    for(i=0;i<n;i++)
    {
      auxX[i]=analogRead(A10)/4;
      auxY[i]=analogRead(A9)/4;
      auxZ[i]=analogRead(A7)/4;
    }
    delay(3500);
    digitalWrite(ledpin,LOW);
    }else
    {
      digitalWrite(ledpin, LOW);
    }

    cuenta = 0;
    }else
    {
      cuenta++;
      for(i=0;i<n;i++)
      {
        if(i==(n-1))
        {
          auxX[i]=analogRead(A10)/4;
          auxY[i]=analogRead(A9)/4;
          auxZ[i]=analogRead(A7)/4;
        }else
        {
          auxX[i]=auxX[i+1];
          auxY[i]=auxY[i+1];
          auxZ[i]=auxZ[i+1];
        }
      }
    }
  delay(16);
}
```

izqlectura_cc_arduino_led.ino

```
#include <avr/pgmspace.h>
#include <stdint.h>
#include <math.h>

int ledpin = 7;

//PATRÓN
const uint16_t patronIzqX[] PROGMEM = {83, 83, 84, 83, 84, 83, 84, 83, 83, 83, 83, 83, 82,
83, 82,
              82, 81, 83, 83, 82, 81, 82, 81, 80, 78, 79, 78, 77, 76, 75,
              75, 72, 70, 70, 68, 66, 66, 66, 66, 65, 64, 61, 60, 61, 63,
              67, 68, 69, 71, 74, 77, 78, 85, 91, 90, 92, 96, 102, 109,
              113, 113, 113, 113, 116, 118, 124, 126, 128, 130, 131, 132,
              134, 139, 138, 143, 145, 151, 156, 162, 167, 167, 164, 155,
              150, 148, 149, 151, 153, 151, 149, 147, 147, 149, 148, 148,
              150, 150, 151, 152, 153, 154, 156, 159, 163, 162, 161, 160,
              157, 161, 159, 157, 157, 155, 149, 145, 140, 138, 137, 135,
              132, 131, 131, 130, 131, 133, 134, 131, 131, 131, 131, 131,
              131, 131, 131, 136, 139, 140, 137, 134, 136, 137, 138, 138,
              141, 140, 141, 143, 141, 140, 141, 141, 138, 140, 150, 129,
              136, 132, 131, 136, 135, 136, 135, 134, 135, 133, 134, 134,
              134, 134, 135, 135, 135, 135, 137, 137, 135, 135, 134, 136,
              135, 135, 136, 135, 136, 136, 135, 134, 134, 134, 134, 134,
              134, 134, 134, 134, 134, 135, 134, 134, 134, 135, 134, 133,
              135, 133, 133, 133, 134, 134, 133, 133, 134, 133, 133, 133,
              133, 133, 133, 132, 134, 132, 134, 133, 133, 133, 132, 133,
              133, 133, 133, 133, 134, 133, 133, 133, 133, 134, 134, 134,
              134};
const uint16_t patronIzqY[] PROGMEM = {129, 128, 129, 129, 128, 129, 129, 128, 129, 129,
130, 130,
              129, 129, 130, 129, 129, 128, 129, 129, 129, 129, 129, 128,
              128, 128, 127, 126, 125, 126, 126, 127, 127, 125, 123, 121,
              123, 123, 120, 122, 121, 116, 113, 113, 115, 119, 120, 125,
              130, 137, 140, 142, 143, 141, 138, 134, 133, 135, 134, 136,
              133, 127, 123, 119, 120, 124, 126, 122, 117, 109, 105, 106,
              105, 105, 101, 98, 93, 87, 81, 73, 57, 51, 58, 72, 78, 75,
              76, 82, 92, 92, 88, 82, 78, 76, 78, 76, 76, 74, 72, 72, 71,
              64, 62, 59, 59, 61, 64, 67, 68, 68, 69, 71, 72, 73, 76, 82,
              88, 97, 101, 107, 109, 112, 117, 122, 127, 126, 129, 133,
              138, 138, 139, 142, 149, 159, 172, 177, 179, 177, 175, 175,
              176, 183, 190, 190, 185, 185, 188, 196, 198, 201, 202, 205,
              205, 212, 211, 188, 206, 219, 218, 209, 202, 204, 207, 204,
              202, 203, 203, 199, 194, 195, 194, 194, 192, 192, 194, 192,
              189, 188, 187, 186, 186, 189, 186, 186, 184, 185, 186, 186,
              186, 186, 186, 187, 186, 186, 186, 187, 186, 187, 188, 187,
              187, 187, 187, 188, 188, 187, 187, 187, 186, 186, 186, 187,
              187, 188, 188, 188, 187, 186, 187, 187, 189, 188, 187, 188,
              188, 188, 186, 187, 188, 189, 188, 188, 187, 187, 187, 188,
              188, 188, 188, 187};
const uint16_t patronIzqZ[] PROGMEM = {120, 120, 121, 120, 120, 119, 120, 120, 120, 120,
121, 121,
              120, 122, 119, 120, 120, 120, 120, 121, 122, 121, 122, 121,
              120, 121, 121, 122, 124, 124, 124, 123, 122, 123, 123, 123,
              127, 128, 131, 132, 134, 137, 140, 142, 142, 143, 144, 142,
              142, 144, 148, 150, 150, 150, 151, 153, 155, 156, 158, 161,
              163, 165, 167, 169, 167, 168, 170, 170, 172, 171, 170, 169,
              171, 169, 173, 172, 172, 171, 173, 175, 177, 173, 170, 172,
              173, 172, 172, 173, 175, 176, 175, 175, 174, 174, 174, 174,
              176, 176, 176, 175, 174, 178, 178, 178, 178, 179, 179, 178,
              180, 180, 182, 181, 179, 178, 176, 175, 176, 175, 173, 172,
              174, 176, 171, 171, 170, 169, 168, 166, 163, 164, 162, 159,
              155, 148, 145, 145, 146, 145, 144, 147, 145, 139, 138, 138,
              138, 134, 134, 131, 128, 126, 123, 123, 122, 121, 153, 127,
              121, 132, 132, 128, 127, 127, 125, 124, 125, 126, 127, 126,
              125, 125, 125, 126, 125, 125, 125, 125, 124, 124, 124, 125,
```

```
              126, 125, 125, 124, 125, 124, 126, 125, 125, 124, 127, 126,
              127, 126, 125, 126, 126, 125, 125, 126, 126, 126, 125, 125,
              126, 126, 126, 125, 126, 126, 125, 126, 126, 126, 126, 126,
              125, 127, 126, 126, 126, 125, 126, 126, 126, 126, 126, 125,
              125, 126, 125, 125, 125, 126, 125, 127, 126, 126, 125, 125};

int i = 0;
int j=0;

int n = 240;
int cuenta = 50;

double varX=0;
double varY=0;
double varZ=0;
uint16_t patronX=0;
uint16_t patronY=0;
uint16_t patronZ=0;
double Xxy=0;
double Yxy=0;
double Zxy=0;
double Xxsquare=0;
double Yxsquare=0;
double Zxsquare=0;
double Xysquare=0;
double Yysquare=0;
double Zysquare=0;
double Xxsum=0;
double Yxsum=0;
double Zxsum=0;
double Xysum=0;
double Yysum=0;
double Zysum = 0;
double Xxysum=0;
double Yxysum=0;
double Zxysum=0;
double Xxsqr_sum=0;
double Yxsqr_sum=0;
double Zxsqr_sum=0;
double Xysqr_sum=0;
double Yysqr_sum=0;
double Zysqr_sum=0;
double numX=0;
double denoX=0;
double numY=0;
double denoY=0;
double numZ=0;
double denoZ=0;
double coeffX=0;
double coeffY=0;
double coeffZ=0;

uint16_t auxX [240];
uint16_t auxY [240];
uint16_t auxZ [240];


void setup() {
  // Inicializar comunicación por puerto serie a 115200 bits por segundo
  pinMode(A10, INPUT);
  pinMode(A9, INPUT);
  pinMode(A7, INPUT);

}

void loop() {
  if (j==0)
  {
    //PRIMERO LLENAMOS LOS VECTORES DE DATOS QUE "RECIBO"
    for(i=0;i<n;i++)
```

```
    {
      auxX[i]=analogRead(A10)/4;
      auxY[i]=analogRead(A9)/4;
      auxZ[i]=analogRead(A7)/4;
    }
    j=1;
  }
  //COMPARACIÓN ENTRE EL VECTOR DE DATOS Y EL PATRÓN CADA 50 MUESTRAS
  if (cuenta==50)
  {
    digitalWrite(ledpin, HIGH);
    delay(1);
    digitalWrite(ledpin, LOW);

    Xxsum=0;
    Xysum=0;
    Xxysum=0;
    Xxsqr_sum=0;
    Xysqr_sum=0;

    Yxsum=0;
    Yysum=0;
    Yxysum=0;
    Yxsqr_sum=0;
    Yysqr_sum=0;

    Zxsum=0;
    Zysum=0;
    Zxysum=0;
    Zxsqr_sum=0;
    Zysqr_sum=0;

    for (i=0; i<n; i++)
    {
      //Lectura del patrón
      patronX=pgm_read_word_near(patronIzqX + i);
      patronY=pgm_read_word_near(patronIzqY + i);
      patronZ=pgm_read_word_near(patronIzqZ + i);

      //Correlacion cruzada eje X
      Xxy= auxX[i] * patronX;
      Xxsquare = auxX[i] * auxX[i];
      Xysquare = patronX * patronX;
      Xxsum = Xxsum + auxX[i];
      Xysum = Xysum + patronX;
      Xxysum = Xxysum + Xxy;
      Xxsqr_sum = Xxsqr_sum + Xxsquare;
      Xysqr_sum = Xysqr_sum + Xysquare;

      //Correlacion cruzada eje Y
      Yxy= auxY[i] * patronY;
      Yxsquare = auxY[i] * auxY[i];
      Yysquare = patronY * patronY;
      Yxsum = Yxsum + auxY[i];
      Yysum = Yysum + patronY;
      Yxysum = Yxysum + Yxy;
      Yxsqr_sum = Yxsqr_sum + Yxsquare;
      Yysqr_sum = Yysqr_sum + Yysquare;

      //Correlacion cruzada eje Z
      Zxy = auxZ[i] * patronZ;
      Zxsquare = auxZ[i] * auxZ[i];
      Zysquare = patronZ * patronZ;
      Zxsum = Zxsum + auxZ[i];
      Zysum = Zysum + patronZ;
      Zxysum = Zxysum + Zxy;
      Zxsqr_sum = Zxsqr_sum + Zxsquare;
      Zysqr_sum = Zysqr_sum + Zysquare;
    }
```

```
    //Coeficiente Correlación Cruzada X: GESTO GIRO IZQUIERDA
    numX=1.0*((n*Xxysum)-(Xxsum*Xysum));
    denoX=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
    coeffX=numX/sqrt(denoX);

    //Coeficiente Correlacion Cruzada Y: GESTO GIRO IZQUIERDA
    numY=1.0*((n*Yxysum)-(Yxsum*Yysum));
    denoY=1.0*((n*Yxsqr_sum-Yxsum*Yxsum)*(n*Yysqr_sum-Yysum*Yysum));
    coeffY=numY/sqrt(denoY);

    //Coeficiente Correlacion Cruzada Z: GESTO GIRO IZQUIERDA
    numZ=((n*Zxysum)-(Zxsum*Zysum));
    denoZ=((n*Zxsqr_sum-Zxsum*Zxsum)*(n*Zysqr_sum-Zysum*Zysum));
    coeffZ=numZ/sqrt(denoZ);

    if (coeffX>0.5 && coeffY>0.5 && coeffZ>0.5)
    {
      digitalWrite(ledpin, HIGH);
      for(i=0;i<n;i++)
      {
        auxX[i]=analogRead(A10)/4;
        auxY[i]=analogRead(A9)/4;
        auxZ[i]=analogRead(A7)/4;
      }
      delay(3500);
      digitalWrite(ledpin,LOW);
    }else
    {
      digitalWrite(ledpin, LOW);
    }
    cuenta = 0;
  }else
  {
    cuenta++;
    for(i=0;i<n;i++)
    {
      if(i==(n-1))
      {
        auxX[i]=analogRead(A10)/4;
        auxY[i]=analogRead(A9)/4;
        auxZ[i]=analogRead(A7)/4;
      }else
      {
        auxX[i]=auxX[i+1];
        auxY[i]=auxY[i+1];
        auxZ[i]=auxZ[i+1];
      }
    }
  }
  delay(16);
}
```

A.6 Código 7. Implementación final en FLORA.

tfg.ino

```
#include <avr/pgmspace.h>
#include <stdint.h>
#include <math.h>

#define NADA 0
#define COMPARA 1
#define LED_DER 2
#define LED_IZQ 3

#define MAX_PARPADEOS 6
```

```
//Patrón de giro a la derecha
const uint16_t GiroDerX[] PROGMEM = {83, 84, 84, 83, 84, 83, 84, 83, 84, 83, 84, 83, 83,
83,
                83, 83, 83, 84, 84, 84, 83, 84, 84, 84, 84, 83, 82, 83,
                84, 82, 81, 80, 78, 76, 74, 72, 66, 66, 66, 62, 60, 58,
                57, 59, 63, 66, 70, 75, 76, 78, 83, 85, 91, 92, 95, 96,
                99, 104, 109, 109, 114, 116, 120, 124, 125, 132, 133, 136,
                134, 141, 142, 143, 147, 150, 151, 151, 153, 149, 157,
                154, 157, 155, 153, 150, 148, 147, 146, 142, 140, 138,
                140, 139, 138, 138, 137, 137, 139, 138, 140, 140, 142,
                141, 140, 140, 141, 142, 143, 143, 142, 141, 140, 142,
                141, 140, 140, 140, 140, 139, 139, 140, 140, 140, 141,
                141, 142, 144, 147, 149, 152, 150, 148, 149, 148, 148,
                147, 147, 148, 145, 142, 141, 138, 137, 136, 134, 137,
                133, 131, 129, 119, 109, 102, 102, 108, 110, 106, 100,
                97, 88, 84, 80, 78, 76, 77, 73, 69, 64, 67, 66, 70, 68,
                68, 64, 65, 68, 67, 52, 59, 79, 86, 88, 88, 84, 83, 85,
                83, 82, 82, 81, 82, 84, 84, 84, 84, 84, 83, 83, 82, 82,
                84, 84, 83, 83, 83, 82, 83, 83, 83, 82, 83, 84, 83, 84,
                85, 84, 84, 82, 82, 83, 84, 83, 84, 83, 84, 83, 82, 82,
                82, 83, 83, 84, 84, 84, 83, 84, 84, 84, 84, 83, 83, 84};
const uint16_t GiroDerY[] PROGMEM = {136, 136, 136, 137, 136, 136, 136, 137, 136, 137, 137,
                136, 136, 136, 136, 137, 136, 138, 137, 137, 136, 136,
                136, 135, 137, 137, 136, 137, 136, 135, 136, 132, 133,
                134, 134, 136, 137, 141, 133, 129, 129, 131, 135, 136,
                137, 138, 140, 140, 141, 142, 143, 143, 142, 139, 139,
                138, 137, 136, 134, 133, 133, 133, 131, 127, 125, 124,
                127, 124, 120, 119, 117, 115, 112, 112, 114, 112, 109,
                109, 108, 108, 111, 112, 112, 110, 110, 112, 114, 114,
                116, 115, 115, 115, 117, 115, 114, 115, 115, 115, 115,
                115, 114, 114, 114, 114, 114, 114, 114, 115, 116, 114,
                114, 114, 114, 114, 114, 116, 116, 115, 116, 115, 114,
                114, 114, 114, 114, 113, 113, 111, 110, 108, 107, 110,
                112, 114, 113, 113, 115, 117, 118, 121, 122, 124, 125,
                125, 125, 128, 132, 133, 130, 129, 133, 137, 138, 139,
                138, 137, 137, 137, 139, 139, 144, 145, 144, 139, 141,
                140, 138, 135, 136, 137, 139, 137, 135, 133, 129, 133,
                142, 138, 135, 136, 133, 136, 137, 138, 138, 140, 139,
                139, 139, 137, 135, 136, 135, 135, 135, 136, 137, 137,
                135, 136, 135, 136, 136, 136, 135, 134, 135, 135, 136,
                137, 138, 138, 138, 137, 135, 136, 135, 135, 137, 136,
                137, 137, 137, 136, 136, 135, 136, 136, 136, 137, 137,
                137, 137, 138, 137, 137, 136, 135, 135, 136};
const uint16_t GiroDerZ[] PROGMEM = {130, 129, 129, 130, 129, 130, 130, 129, 129, 130, 129,
                130, 129, 129, 129, 130, 129, 129, 129, 129, 130, 129,
                130, 130, 130, 130, 129, 130, 131, 130, 130, 130, 131,
                129, 127, 128, 124, 122, 128, 130, 125, 125, 129, 133,
                129, 127, 132, 135, 133, 134, 135, 137, 139, 142, 144,
                146, 149, 154, 158, 161, 164, 167, 171, 173, 174, 179,
                182, 183, 183, 186, 188, 188, 191, 192, 193, 193, 197,
                194, 196, 194, 196, 196, 193, 194, 194, 193, 193, 192,
                192, 193, 192, 192, 192, 192, 192, 191, 193, 192, 193,
                192, 193, 192, 192, 193, 193, 193, 193, 193, 193, 193,
                193, 193, 193, 192, 193, 192, 193, 193, 192, 192, 192,
                193, 193, 193, 194, 194, 194, 195, 195, 197, 195, 193,
                193, 194, 192, 193, 191, 190, 189, 189, 186, 185, 184,
                184, 181, 179, 175, 171, 169, 163, 163, 164, 167, 162,
                151, 146, 147, 147, 142, 138, 134, 132, 131, 131, 128,
                124, 123, 129, 131, 126, 127, 129, 130, 129, 128, 122,
                125, 130, 130, 128, 130, 130, 130, 129, 129, 130, 130,
                129, 130, 131, 130, 129, 130, 130, 130, 130, 129, 129,
                129, 129, 130, 130, 130, 130, 130, 129, 130, 130, 130,
                129, 130, 129, 130, 130, 130, 130, 129, 129, 129, 129,
                130, 129, 129, 129, 129, 130, 130, 130, 130, 129, 130,
                129, 129, 130, 129, 129, 129, 129, 129, 129};

//Patrón de giro a la izquierda
```

```
const uint16_t GiroIzqX[] PROGMEM = {83, 83, 84, 83, 84, 83, 84, 83, 83, 83, 83, 83, 82,
83, 82,
              82, 81, 83, 83, 82, 81, 82, 81, 80, 78, 79, 78, 77, 76, 75,
              75, 72, 70, 70, 68, 66, 66, 66, 66, 65, 64, 61, 60, 61, 63,
              67, 68, 69, 71, 74, 77, 78, 85, 91, 90, 92, 96, 102, 109,
              113, 113, 113, 113, 116, 118, 124, 126, 128, 130, 131, 132,
              134, 139, 138, 143, 145, 151, 156, 162, 167, 167, 164, 155,
              150, 148, 149, 151, 153, 151, 149, 147, 147, 149, 148, 148,
              150, 150, 151, 152, 153, 154, 156, 159, 163, 162, 161, 160,
              157, 161, 159, 157, 157, 155, 149, 145, 140, 138, 137, 135,
              132, 131, 131, 130, 131, 133, 134, 131, 131, 131, 131, 131,
              131, 131, 131, 136, 139, 140, 137, 134, 136, 137, 138, 138,
              141, 140, 141, 143, 141, 140, 141, 141, 138, 140, 150, 129,
              136, 132, 131, 136, 135, 136, 135, 134, 135, 133, 134, 134,
              134, 134, 135, 135, 135, 135, 137, 137, 135, 135, 134, 136,
              135, 135, 136, 135, 136, 136, 135, 134, 134, 134, 134, 134,
              134, 134, 134, 134, 134, 135, 134, 134, 134, 135, 134, 133,
              135, 133, 133, 133, 134, 134, 133, 133, 134, 133, 133, 133,
              133, 133, 133, 133, 132, 134, 132, 134, 133, 133, 133, 132, 133,
              133, 133, 133, 133, 134, 133, 133, 133, 133, 134, 134, 134,
              134};
const uint16_t GiroIzqY[] PROGMEM = {129, 128, 129, 129, 128, 129, 129, 128, 129, 129, 130,
130,
              129, 129, 130, 129, 129, 128, 129, 129, 129, 129, 129, 128,
              128, 128, 127, 126, 125, 126, 126, 127, 127, 125, 123, 121,
              123, 123, 120, 122, 121, 116, 113, 113, 115, 119, 120, 125,
              130, 137, 140, 142, 143, 141, 138, 134, 133, 135, 134, 136,
              133, 127, 123, 119, 120, 124, 126, 122, 117, 109, 105, 106,
              105, 105, 101, 98, 93, 87, 81, 73, 57, 51, 58, 72, 78, 75,
              76, 82, 92, 92, 88, 82, 78, 76, 78, 76, 76, 74, 72, 72, 71,
              64, 62, 59, 59, 61, 64, 67, 68, 68, 69, 71, 72, 73, 76, 82,
              88, 97, 101, 107, 109, 112, 117, 122, 127, 126, 129, 133,
              138, 138, 139, 142, 149, 159, 172, 177, 179, 177, 175, 175,
              176, 183, 190, 190, 185, 185, 188, 196, 198, 201, 202, 205,
              205, 212, 211, 188, 206, 219, 218, 209, 202, 204, 207, 204,
              202, 203, 203, 199, 194, 195, 194, 194, 192, 192, 194, 192,
              189, 188, 187, 186, 186, 189, 186, 186, 184, 185, 186, 186,
              186, 186, 186, 187, 186, 186, 186, 187, 186, 187, 188, 187,
              187, 187, 187, 188, 188, 187, 187, 187, 186, 186, 186, 187,
              187, 188, 188, 188, 187, 186, 187, 187, 189, 188, 187, 188,
              188, 188, 186, 187, 188, 189, 188, 188, 187, 187, 187, 188,
              188, 188, 188, 187};
const uint16_t GiroIzqZ[] PROGMEM = {120, 120, 121, 120, 120, 119, 120, 120, 120, 120, 121,
121,
              120, 122, 119, 120, 120, 120, 120, 121, 122, 121, 122, 121,
              120, 121, 121, 122, 124, 124, 124, 123, 122, 123, 123, 123,
              127, 128, 131, 132, 134, 137, 140, 142, 142, 143, 144, 142,
              142, 144, 148, 150, 150, 150, 151, 153, 155, 156, 158, 161,
              163, 165, 167, 169, 167, 168, 170, 170, 172, 171, 170, 169,
              171, 169, 173, 172, 172, 171, 173, 175, 177, 173, 170, 172,
              173, 172, 172, 173, 175, 176, 175, 175, 174, 174, 174, 174,
              176, 176, 176, 175, 174, 178, 178, 178, 178, 179, 179, 178,
              180, 180, 182, 181, 179, 178, 176, 175, 176, 175, 173, 172,
              174, 176, 171, 171, 170, 169, 168, 166, 163, 164, 162, 159,
              155, 148, 145, 145, 146, 145, 144, 147, 145, 139, 138, 138,
              138, 134, 134, 131, 128, 126, 123, 123, 122, 121, 153, 127,
              121, 132, 132, 128, 127, 127, 125, 124, 125, 126, 127, 126,
              125, 125, 125, 126, 125, 125, 125, 125, 124, 124, 124, 125,
              126, 125, 125, 124, 125, 124, 126, 125, 125, 124, 127, 126,
              127, 126, 125, 126, 126, 125, 125, 126, 126, 126, 125, 125,
              126, 126, 126, 125, 126, 126, 125, 126, 126, 126, 126, 126,
              125, 127, 126, 126, 126, 125, 126, 126, 126, 126, 126, 125,
              125, 126, 125, 125, 125, 126, 125, 127, 126, 126, 125, 125};


int ledpin = 7;

int i = 0;
int j = 0;
```

```
int n = 240;
int cuenta = 40;

int estado=COMPARA;

double varX=0;
double varY=0;
double varZ=0;

double DcoeffX;
double DcoeffY;
double DcoeffZ;
double IcoeffX;
double IcoeffY;
double IcoeffZ;

//Variables para leer datos del acelerómetro
uint16_t auxX [240];
uint16_t auxY [240];
uint16_t auxZ [240];

const int led_der1  =  0;
const int led_izq1 = 2;
int cuenta_parpadeos=0;

void setup() {
  pinMode(led_der1,OUTPUT);
  pinMode(led_izq1,OUTPUT);

  pinMode(A10, INPUT);
  pinMode(A9, INPUT);
  pinMode(A7, INPUT);
}

void loop() {

  switch (estado)
  {
    case NADA:
      muestreo(&cuenta, auxX, auxY, auxZ, n);
      cuenta++;
      if(cuenta==40)
      {
        estado=COMPARA;
      }
      break;
    case COMPARA:
      if (j==0)
      {
        lee_datos(auxX, auxY, auxZ,n);
        j=1;
      }
      compara_patron(auxX, auxY, auxZ, &DcoeffX, &DcoeffY, &DcoeffZ, &IcoeffX, &IcoeffY,
&IcoeffZ, ledpin, n);
      if (DcoeffX>0.6 && DcoeffY>0.6 && DcoeffZ>0.6)
      {
        estado=LED_DER;
        cuenta=0;
      }else if (IcoeffX>0.6 && IcoeffY>0.6 && IcoeffZ>0.6)
      {
        estado=LED_IZQ;
        cuenta=0;
      }else
      {
        estado=NADA;
        cuenta=0;
      }
      break;
    case LED_DER:
```

```
      digitalWrite(led_der1,HIGH);
      delay(500);
      digitalWrite(led_der1,LOW);
      delay(500);
      cuenta_parpadeos++;
      if(cuenta_parpadeos >= (MAX_PARPADEOS))
      {
        estado=NADA;
        cuenta_parpadeos=0;
        lee_datos(auxX, auxY, auxZ,n);
      }
      break;
    case LED_IZQ:
      digitalWrite(led_izq1,HIGH);
      delay(500);
      digitalWrite(led_izq1,LOW);
      delay(500);
      cuenta_parpadeos++;
      if(cuenta_parpadeos >= MAX_PARPADEOS)
      {
        estado=NADA;
        cuenta_parpadeos=0;
        lee_datos(auxX, auxY, auxZ,n);
      }
      break;
    default:
      break;
  }
}

void muestreo(int*cuenta, uint16_t * auxX,uint16_t * auxY, uint16_t * auxZ, int n){

  int i=0;
  *cuenta=(*cuenta)+1;
  for(i=0;i<n;i++)
  {
    if(i==(n-1))
    {
      *(auxX+i)=analogRead(A10)/4;
      *(auxY+i)=analogRead(A9)/4;
      *(auxZ+i)=analogRead(A7)/4;
    }else
    {
      *(auxX+i)=*(auxX+(i+1));
      *(auxY+i)=*(auxY+(i+1));
      *(auxZ+i)=*(auxZ+(i+1));
    }
  }
}


void  compara_patron(uint16_t  *auxX,  uint16_t*auxY,  uint16_t*auxZ,  double*DcoeffX,
double*DcoeffY, double*DcoeffZ, double*IcoeffX, double*IcoeffY, double*IcoeffZ, int ledpin,
int n){

  int i=0;
  //Variables lectura patron derecho
  uint16_t DpatronX=0;
  uint16_t DpatronY=0;
  uint16_t DpatronZ=0;

  //Variables lectura patron izquierdo
  uint16_t IpatronX=0;
  uint16_t IpatronY=0;
  uint16_t IpatronZ=0;

  double Xxy=0;
  double Yxy=0;
  double Zxy=0;
  double Xxsquare=0;
```

```
double Yxsquare=0;
double Zxsquare=0;
double Xysquare=0;
double Yysquare=0;
double Zysquare=0;
double Xxsum=0;
double Yxsum=0;
double Zxsum=0;
double Xysum=0;
double Yysum=0;
double Zysum = 0;
double Xxysum=0;
double Yxysum=0;
double Zxysum=0;
double Xxsqr_sum=0;
double Yxsqr_sum=0;
double Zxsqr_sum=0;
double Xysqr_sum=0;
double Yysqr_sum=0;
double Zysqr_sum=0;

//Variables asociadas al gesto derecho
double DnumX=0;
double DdenoX=0;
double DnumY=0;
double DdenoY=0;
double DnumZ=0;
double DdenoZ=0;

//Variables asociadas al gesto izquierdo
double InumX=0;
double IdenoX=0;
double InumY=0;
double IdenoY=0;
double InumZ=0;
double IdenoZ=0;

digitalWrite(ledpin, HIGH);
delay(1);
digitalWrite(ledpin, LOW);

Xxsum=0;
Xysum=0;
Xxysum=0;
Xxsqr_sum=0;
Xysqr_sum=0;

Yxsum=0;
Yysum=0;
Yxysum=0;
Yxsqr_sum=0;
Yysqr_sum=0;

Zxsum=0;
Zysum=0;
Zxysum=0;
Zxsqr_sum=0;
Zysqr_sum=0;

// Primero comparamos lo recibido con el patrón derecho.
for (i=0; i<n; i++)
{
  //Lectura del patrón derecho
  DpatronX=pgm_read_word_near(GiroDerX + i);
  DpatronY=pgm_read_word_near(GiroDerY + i);
  DpatronZ=pgm_read_word_near(GiroDerZ + i);

  //Correlacion cruzada eje X
  Xxy= auxX[i] * DpatronX;
```

```
  Xxsquare = auxX[i] * auxX[i];
  Xysquare = DpatronX * DpatronX;
  Xxsum = Xxsum + auxX[i];
  Xysum = Xysum + DpatronX;
  Xxysum = Xxysum + Xxy;
  Xxsqr_sum = Xxsqr_sum + Xxsquare;
  Xysqr_sum = Xysqr_sum + Xysquare;

  //Correlacion cruzada eje Y
  Yxy= auxY[i] * DpatronY;
  Yxsquare = auxY[i] * auxY[i];
  Yysquare = DpatronY * DpatronY;
  Yxsum = Yxsum + auxY[i];
  Yysum = Yysum + DpatronY;
  Yxysum = Yxysum + Yxy;
  Yxsqr_sum = Yxsqr_sum + Yxsquare;
  Yysqr_sum = Yysqr_sum + Yysquare;

  //Correlacion cruzada eje Z
  Zxy = auxZ[i] * DpatronZ;
  Zxsquare = auxZ[i] * auxZ[i];
  Zysquare = DpatronZ * DpatronZ;
  Zxsum = Zxsum + auxZ[i];
  Zysum = Zysum + DpatronZ;
  Zxysum = Zxysum + Zxy;
  Zxsqr_sum = Zxsqr_sum + Zxsquare;
  Zysqr_sum = Zysqr_sum + Zysquare;
}

//Coeficiente Correlación Cruzada X: GESTO GIRO DERECHA
DnumX=1.0*((n*Xxysum)-(Xxsum*Xysum));
DdenoX=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
(*DcoeffX)=DnumX/sqrt(DdenoX);

//Coeficiente Correlacion Cruzada Y: GESTO GIRO DERECHA
DnumY=1.0*((n*Yxysum)-(Yxsum*Yysum));
DdenoY=1.0*((n*Yxsqr_sum-Yxsum*Yxsum)*(n*Yysqr_sum-Yysum*Yysum));
(*DcoeffY)=DnumY/sqrt(DdenoY);

//Coeficiente Correlacion Cruzada Z: GESTO GIRO DERECHA
DnumZ=((n*Zxysum)-(Zxsum*Zysum));
DdenoZ=((n*Zxsqr_sum-Zxsum*Zxsum)*(n*Zysqr_sum-Zysum*Zysum));
(*DcoeffZ)=DnumZ/sqrt(DdenoZ);

Xxsum=0;
Xysum=0;
Xxysum=0;
Xxsqr_sum=0;
Xysqr_sum=0;

Yxsum=0;
Yysum=0;
Yxysum=0;
Yxsqr_sum=0;
Yysqr_sum=0;

Zxsum=0;
Zysum=0;
Zxysum=0;
Zxsqr_sum=0;
Zysqr_sum=0;

for (i=0; i<n; i++)
{
  //Lectura del patrón izquierdo
  IpatronX=pgm_read_word_near(GiroIzqX + i);
  IpatronY=pgm_read_word_near(GiroIzqY + i);
  IpatronZ=pgm_read_word_near(GiroIzqZ + i);
```

```
    //Correlacion cruzada eje X
    Xxy= auxX[i] * IpatronX;
    Xxsquare = auxX[i] * auxX[i];
    Xysquare = IpatronX * IpatronX;
    Xxsum = Xxsum + auxX[i];
    Xysum = Xysum + IpatronX;
    Xxysum = Xxysum + Xxy;
    Xxsqr_sum = Xxsqr_sum + Xxsquare;
    Xysqr_sum = Xysqr_sum + Xysquare;

    //Correlacion cruzada eje Y
    Yxy= auxY[i] * IpatronY;
    Yxsquare = auxY[i] * auxY[i];
    Yysquare = IpatronY * IpatronY;
    Yxsum = Yxsum + auxY[i];
    Yysum = Yysum + IpatronY;
    Yxysum = Yxysum + Yxy;
    Yxsqr_sum = Yxsqr_sum + Yxsquare;
    Yysqr_sum = Yysqr_sum + Yysquare;

    //Correlacion cruzada eje Z
    Zxy = auxZ[i] * IpatronZ;
    Zxsquare = auxZ[i] * auxZ[i];
    Zysquare = IpatronZ * IpatronZ;
    Zxsum = Zxsum + auxZ[i];
    Zysum = Zysum + IpatronZ;
    Zxysum = Zxysum + Zxy;
    Zxsqr_sum = Zxsqr_sum + Zxsquare;
    Zysqr_sum = Zysqr_sum + Zysquare;
  }

  //Coeficiente Correlación Cruzada X: GESTO GIRO IZQUIERDA
  InumX=1.0*((n*Xxysum)-(Xxsum*Xysum));
  IdenoX=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
  (*IcoeffX)=InumX/sqrt(IdenoX);

  //Coeficiente Correlacion Cruzada Y: GESTO GIRO IZQUIERDA
  InumY=1.0*((n*Yxysum)-(Yxsum*Yysum));
  IdenoY=1.0*((n*Yxsqr_sum-Yxsum*Yxsum)*(n*Yysqr_sum-Yysum*Yysum));
  (*IcoeffY)=InumY/sqrt(IdenoY);

  //Coeficiente Correlacion Cruzada Z: GESTO GIRO IZQUIERDA
  InumZ=((n*Zxysum)-(Zxsum*Zysum));
  IdenoZ=((n*Zxsqr_sum-Zxsum*Zxsum)*(n*Zysqr_sum-Zysum*Zysum));
  (*IcoeffZ)=InumZ/sqrt(IdenoZ);
}

void lee_datos(uint16_t * auxX, uint16_t * auxY, uint16_t * auxZ, int n){
  int i=0;
  for(i=0;i<n;i++)
  {
    *(auxX+i)=analogRead(A10)/4;
    *(auxY+i)=analogRead(A9)/4;
    *(auxZ+i)=analogRead(A7)/4;
  }
}
```

ANEXO B. CÓDIGOS MATLAB®

B.1 Código 1. Captura de un gesto.

script_captura_patron.m

```
close all
clc
%Apertura de conexión: Puerto COM4. Velocidad de Tx: 115200bps
a = serial('COM4', 'BaudRate', 115200);
fopen(a);

disp('Tomando datos...');
bp=waitbar(0,'Espere por favor...');
total=1200;
dataX=zeros(total,1);
dataY=zeros(total,1);
dataZ=zeros(total,1);
time=zeros(total,1);
exit=0;
k=[1:5:1200];
fprintf(a,'1');
for i=1:total,
    dataX(i,1)=str2double(fscanf(a));
    dataY(i,1)=str2double(fscanf(a));
    dataZ(i,1)=str2double(fscanf(a));
    time(i,1)=str2double(fscanf(a));
    waitbar(i/total,bp); %Actualizando la barra de progreso
end
fprintf(a,'0');
x=(time-time(1))/1000;

disp('Fin de la toma de datos');
fclose(a);
waitbar(100,bp,'Completado');
pause(1)
close(bp);

%Redondear y reducir de 1200 muestras a 240
data2X=round(dataX(k,1)/4,0);
data2Y=round(dataY(k,1)/4,0);
data2Z=round(dataZ(k,1)/4,0);
x2=x(k,1);
figure(1)
plot(x2,data2X)
figure(2)
plot(x2,data2Y)
figure(3)
plot(x2,data2Z)

delete(a)
```

B.2 Código 2. Identificación del gesto. Cálculo del parecido.

script_cc_matlab_vs_c.m

```
clear
clc
load gesto.mat
load patron.mat

n=240;

%Variables a utilizar por el algoritmo alternativo
Xxsquare=zeros(240,1);
Xysquare=zeros(240,1);
Xxy=zeros(1,240);
Xxsum=0;
Xysum=0;
Xxysum=0;
Xxsqr_sum=0;
Xysqr_sum=0;

Yxsquare=zeros(240,1);
Yysquare=zeros(240,1);
Yxy=zeros(1,240);
Yxsum=0;
Yysum=0;
Yxysum=0;
Yxsqr_sum=0;
Yysqr_sum=0;

Zxsquare=zeros(240,1);
Zysquare=zeros(240,1);
Zxy=zeros(1,240);
Zxsum=0;
Zysum=0;
Zxysum=0;
Zxsqr_sum=0;
Zysqr_sum=0;

%Correlacion usando metodo alternativo
%Comienzo del bucle. En el se resuelven los sumatorios.
for i=1:n,
    Xxy(i)=gestoDerX(i)*patronDerX(i);
    Xxsquare(i)=gestoDerX(i)^2;
    Xysquare(i)=patronDerX(i)^2;
    Xxsum=Xxsum+gestoDerX(i);
    Xysum=Xysum+patronDerX(i);
    Xxysum=Xxysum+Xxy(i);
    Xxsqr_sum=Xxsqr_sum+Xxsquare(i);
    Xysqr_sum=Xysqr_sum+Xysquare(i);
end
%Finalmente se calculan numerador y denominador, y se calcula el coeficiente.
Xnum=1.0*((n*Xxysum)-(Xxsum*Xysum));
Xdeno=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
coeffXMatlab=Xnum/sqrt( Xdeno)

%Correlacion usando corr2
A=corr2(patronDerX,gestoDerX)
```

```
%Correlacion usando metodo alternativo
for j=1:n,
    Yxy(j)=gestoDerY(j)*patronDerY(j);
    Yxsquare(j)=gestoDerY(j)^2;
    Yysquare(j)=patronDerY(j)^2;
    Yxsum=Yxsum+gestoDerY(j);
    Yysum=Yysum+patronDerY(j);
    Yxysum=Yxysum+Yxy(j);
    Yxsqr_sum=Yxsqr_sum+Yxsquare(j);
    Yysqr_sum=Yysqr_sum+Yysquare(j);
end

Ynum=1.0*((n*Yxysum)-(Yxsum*Yysum));
Ydeno=1.0*((n*Yxsqr_sum-Yxsum*Yxsum)*(n*Yysqr_sum-Yysum*Yysum));
coeffYMatlab=Ynum/sqrt(Ydeno)

%Correlacion usando corr2
B=corr2(patronDerY,gestoDerY)


%Correlacion usando metodo alternativo
for l=1:n,
    Zxy(l)=gestoDerZ(l)*patronDerZ(l);
    Zxsquare(l)=gestoDerZ(l)^2;
    Zysquare(l)=patronDerZ(l)^2;
    Zxsum=Zxsum+gestoDerZ(l);
    Zysum=Zysum+patronDerZ(l);
    Zxysum=Zxysum+Zxy(l);
    Zxsqr_sum=Zxsqr_sum+Zxsquare(l);
    Zysqr_sum=Zysqr_sum+Zysquare(l);
end

Znum=1.0*((n*Zxysum)-(Zxsum*Zysum));
Zdeno=1.0*((n*Zxsqr_sum-Zxsum*Zxsum)*(n*Zysqr_sum-Zysum*Zysum));
coeffZMatlab=Znum/sqrt(Zdeno)

%Correlacion usando corr2
C=corr2(patronDerZ,gestoDerZ)
```

script_cc_izq_matlab_vs_c.m

```
clear
clc
load gestoIzq.mat
load patronIzq.mat

n=240;

%Variables a utilizar por el algoritmo alternativo
Xxsquare=zeros(240,1);
Xysquare=zeros(240,1);
Xxy=zeros(1,240);
Xxsum=0;
Xysum=0;
Xxysum=0;
Xxsqr_sum=0;
```

```
Xysqr_sum=0;

Yxsquare=zeros(240,1);
Yysquare=zeros(240,1);
Yxy=zeros(1,240);
Yxsum=0;
Yysum=0;
Yxysum=0;
Yxsqr_sum=0;
Yysqr_sum=0;

Zxsquare=zeros(240,1);
Zysquare=zeros(240,1);
Zxy=zeros(1,240);
Zxsum=0;
Zysum=0;
Zxysum=0;
Zxsqr_sum=0;
Zysqr_sum=0;

%Correlacion usando metodo alternativo
%Comienzo del bucle. En el se resuelven los sumatorios.
for i=1:n,
    Xxy(i)=patronIzqX(i)*gestoIzqX(i);
    Xxsquare(i)=patronIzqX(i)^2;
    Xysquare(i)=gestoIzqX(i)^2;
    Xxsum=Xxsum+patronIzqX(i);
    Xysum=Xysum+gestoIzqX(i);
    Xxysum=Xxysum+Xxy(i);
    Xxsqr_sum=Xxsqr_sum+Xxsquare(i);
    Xysqr_sum=Xysqr_sum+Xysquare(i);
end
%Finalmente se calculan numerador y denominador, y se calcula el coeficiente.
Xnum=1.0*((n*Xxysum)-(Xxsum*Xysum));
Xdeno=1.0*((n*Xxsqr_sum-Xxsum*Xxsum)*(n*Xysqr_sum-Xysum*Xysum));
coeffXMatlab=Xnum/sqrt(Xdeno)

%Correlacion usando corr2
A=corr2(patronIzqX,gestoIzqX)

%Correlacion usando metodo alternativo
for j=1:n,
    Yxy(j)=patronIzqY(j)*gestoIzqY(j);
    Yxsquare(j)=patronIzqY(j)^2;
    Yysquare(j)=gestoIzqY(j)^2;
    Yxsum=Yxsum+patronIzqY(j);
    Yysum=Yysum+gestoIzqY(j);
    Yxysum=Yxysum+Yxy(j);
    Yxsqr_sum=Yxsqr_sum+Yxsquare(j);
    Yysqr_sum=Yysqr_sum+Yysquare(j);
end
Ynum=1.0*((n*Yxysum)-(Yxsum*Yysum));
Ydeno=1.0*((n*Yxsqr_sum-Yxsum*Yxsum)*(n*Yysqr_sum-Yysum*Yysum));
coeffYMatlab=Ynum/sqrt(Ydeno)

%Correlacion usando corr2
B=corr2(patronIzqY,gestoIzqY)

%Correlacion usando metodo alternativo
for l=1:n,
    Zxy(l)=patronIzqZ(l)*gestoIzqZ(l);
```

```
    Zxsquare(l)=patronIzqZ(l)^2;
    Zysquare(l)=gestoIzqZ(l)^2;
    Zxsum=Zxsum+patronIzqZ(l);
    Zysum=Zysum+gestoIzqZ(l);
    Zxysum=Zxysum+Zxy(l);
    Zxsqr_sum=Zxsqr_sum+Zxsquare(l);
    Zysqr_sum=Zysqr_sum+Zysquare(l);
end

Znum=1.0*((n*Zxysum)-(Zxsum*Zysum));
Zdeno=1.0*((n*Zxsqr_sum-Zxsum*Zxsum)*(n*Zysqr_sum-Zysum*Zysum));
coeffZMatlab=Znum/sqrt(Zdeno)

%Correlacion usando corr2
C=corr2(patronIzqZ,gestoIzqZ)
```

B.3 Código 3. Correlación cruzada en FLORA.

script_cc_arduino.m

```
close all
clc
clear
a = serial('COM4', 'BaudRate', 115200);
fopen(a);
disp(' ')
disp('Tomando datos...');
bp=waitbar(0,'Espere por favor...');

total=240;
flag=0;
fprintf(a,'1');
while (flag~=2)
    flag=str2double(fscanf(a));
end
CCDXArduino=str2double(fscanf(a));
CCDYArduino=str2double(fscanf(a));
CCDZArduino=str2double(fscanf(a));
flag=str2double(fscanf(a));

if(flag==3)
    disp(' ')
    disp('Fin de la toma de datos');
    fclose(a);
    waitbar(100,bp,'Completado');
    disp(' ')
    disp('Corr. Cruzada - Giro Derecha X:')
    disp(CCDXArduino);
    disp('Corr. Cruzada - Giro Derecha Y:')
    disp(CCDYArduino);
    disp('Corr. Cruzada - Giro Derecha Z:')
    disp(CCDZArduino);
end
pause(1)
close(bp);
```

script_cc_arduino_izq.m

```
close all
clc
```

```
clear
a = serial('COM4', 'BaudRate', 115200);
fopen(a);
disp(' ')
disp('Tomando datos...');
bp=waitbar(0,'Espere por favor...');

total=240;
flag=0;
fprintf(a,'1');
while (flag~=2)
    flag=str2double(fscanf(a));
end
CCDXArduino=str2double(fscanf(a));
CCDYArduino=str2double(fscanf(a));
CCDZArduino=str2double(fscanf(a));
flag=str2double(fscanf(a));

if(flag==3)
    disp(' ')
    disp('Fin de la toma de datos');
    fclose(a);
    waitbar(100,bp,'Completado');
    disp(' ')
    disp('Corr. Cruzada - Giro Izquierda X:')
    disp(CCDXArduino);
    disp('Corr. Cruzada - Giro Izquierda Y:')
    disp(CCDYArduino);
    disp('Corr. Cruzada - Giro Izquierda Z:')
    disp(CCDZArduino);
end
pause(1)
close(bp);
```

B.4 Código 4. Probando el desplazamiento.

script_despl_cc_arduino.m

```
close all
clc
a = serial('COM4', 'BaudRate', 115200);
fopen(a);
disp(' ')
disp('Tomando datos...');
bp=waitbar(0,'Espere por favor...');
auxX=ones(1,10);
auxY=ones(1,10);
auxZ=ones(1,10);
total=240;
flag=0;
k=0;
for j=1:6,
    fprintf(a,'1');
    flag=str2double(fscanf(a));
    CCDXArduino=str2double(fscanf(a));
    CCDYArduino=str2double(fscanf(a));
    CCDZArduino=str2double(fscanf(a));

    for r=1:10,
        auxX(1,r)=str2double(fscanf(a));
```

```
    end
    for r=1:10,
        auxY(1,r)=str2double(fscanf(a));
    end
    for r=1:10,
        auxZ(1,r)=str2double(fscanf(a));
    end

    flag=str2double(fscanf(a));

    if(flag==3)
        k=k+1;
        disp(' ')
        disp(['Fin de la toma de datos ', num2str(k), ':']);
        disp(' ')
        disp('    Corr. Cruzada - Giro Derecha X:')
        disp(['    ', num2str(CCDXArduino)]);
        disp('    Corr. Cruzada - Giro Derecha Y:')
        disp(['    ', num2str(CCDYArduino)]);
        disp('    Corr. Cruzada - Giro Derecha Z:')
        disp(['    ', num2str(CCDZArduino)]);
        disp(['   Gesto X (10 últimos datos):'])
        disp(['    [', num2str(auxX),']']);
        disp(['   Gesto Y (10 últimos datos):'])
        disp(['    [', num2str(auxY),']']);
        disp(['   Gesto Z (10 últimos datos):'])
        disp(['    [', num2str(auxZ),']']);
        disp(' ')
        disp(' ')
    end
    for i=0:2,
        fprintf(a,'1');
    end
end
fclose(a);
waitbar(100,bp,'Completado');
close(bp);
```

B.5 Código 5. Captura de gesto, correlación en FLORA y envío de datos a MATLAB®

script_lectura_cc_arduino.m

```
close all
clc
a = serial('COM4', 'BaudRate', 115200);
fopen(a);
disp(' ')
disp('Tomando datos...');
bp=waitbar(0,'Espere por favor...');
auxX=ones(1,10);
auxY=ones(1,10);
auxZ=ones(1,10);
total=240;
flag=0;
k=0;
for j=1:10,
    fprintf(a,'1');
    flag=str2double(fscanf(a));
```

```
    CCDXArduino=str2double(fscanf(a));
    CCDYArduino=str2double(fscanf(a));
    CCDZArduino=str2double(fscanf(a));

    flag=str2double(fscanf(a));

    if(flag==3)
        k=k+1;
        disp(' ')
        disp(['Fin de la toma de datos ', num2str(k), ':']);
        disp(' ')
        disp('    Corr. Cruzada - Giro Derecha X:')
        disp(['     ', num2str(CCDXArduino)]);
        disp('    Corr. Cruzada - Giro Derecha Y:')
        disp(['     ', num2str(CCDYArduino)]);
        disp('    Corr. Cruzada - Giro Derecha Z:')
        disp(['     ', num2str(CCDZArduino)]);
        disp(' ')
        disp(' ')
    end
    for i=0:20,
        fprintf(a,'1');
    end
end
fclose(a);
waitbar(100,bp,'Completado');
close(bp);
```

B.6 Apartado 6.2 – Script para el cálculo de las correlaciones

```
load gesto.mat
load patron.mat
load gestoIzq.mat
load patronIzq.mat

clc

%Correlación Gesto Izquierdo VS Patrón Izquierdo
CorrX_II=corr2(patronIzqX,gestoIzqX);
CorrY_II=corr2(patronIzqY,gestoIzqY);
CorrZ_II=corr2(patronIzqZ,gestoIzqZ);

%Correlación Gesto Izquierdo VS Patrón Derecho
CorrX_ID=corr2(patronDerX,gestoIzqX);
CorrY_ID=corr2(patronDerY,gestoIzqY);
CorrZ_ID=corr2(patronDerZ,gestoIzqZ);

%Correlación Gesto Derecho VS Patrón Izquierdo
CorrX_DI=corr2(patronIzqX,gestoDerX);
CorrY_DI=corr2(patronIzqY,gestoDerY);
CorrZ_DI=corr2(patronIzqZ,gestoDerZ);

%Correlación Gesto Derecho VS Patrón Derecho
CorrX_DD=corr2(patronDerX,gestoDerX);
CorrY_DD=corr2(patronDerY,gestoDerY);
CorrZ_DD=corr2(patronDerZ,gestoDerZ);
disp(' ')
```

```
disp('=== Gestos VS Patrones ===')
disp(' ')
disp('Correlacion Gesto Izquierdo VS Patron Izquierdo')
disp(['     Eje X','     Eje Y','     Eje Z'])
disp([CorrX_II, CorrY_II, CorrZ_II])
disp('Correlacion Gesto Izquierdo VS Patron Derecho')
disp(['     Eje X','     Eje Y','     Eje Z'])
disp([CorrX_ID, CorrY_ID, CorrZ_ID])
disp('Correlacion Gesto Derecho VS Patron Izquierdo')
disp(['     Eje X','     Eje Y','     Eje Z'])
disp([CorrX_DI, CorrY_DI, CorrZ_DI])
disp('Correlacion Gesto Derecho VS Patron Derecho')
disp(['     Eje X','     Eje Y','     Eje Z'])
disp([CorrX_DD, CorrY_DD, CorrZ_DD])
```

C.1 Acelerómetro MMA7260Q

Freescale Semiconductor
Technical Data

MMA7260Q
Rev 0, 04/2005

±1.5g - 6g Three Axis Low-g Micromachined Accelerometer

MMA7260Q

MMA7260Q: XYZ AXIS ACCELEROMETER ±1.5g/2g/4g/6g

The MMA7260Q low cost capacitive micromachined accelerometer features signal conditioning, a 1-pole low pass filter, temperature compensation and g-Select which allows for the selection among 4 sensitivities. Zero-g offset full scale span and filter cut-off are factory set and require no external devices. Includes a Sleep Mode that makes it ideal for handheld battery powered electronics.

Features

- Selectable Sensitivity (1.5g/2g/4g/6g)
- Low Current Consumption: 500 µA
- Sleep Mode: 3 µA
- Low Voltage Operation: 2.2 V – 3.6 V
- 6mm x 6mm x 1.45mm QFN
- High Sensitivity (800 mV/g @1.5 g)
- Fast Turn On Time
- High Sensitivity (1.5 g)
- Integral Signal Conditioning with Low Pass Filter
- Robust Design, High Shocks Survivability
- Pb-Free Terminations
- Environmentally Preferred Package
- Low Cost

Bottom View

16 LEAD QFN CASE 1622-01

Typical Applications

- HDD MP3 Player : Freefall Detection
- Laptop PC : Freefall Detection, Anti-Theft
- Cell Phone : Image Stability, Text Scroll, Motion Dialing, E-Compass
- Pedometer : Motion Sensing
- PDA : Text Scroll
- Navigation and Dead Reckoning : E-Compass Tilt Compensation
- Gaming : Tilt and Motion Sensing, Event Recorder
- Robotics : Motion Sensing

Top View

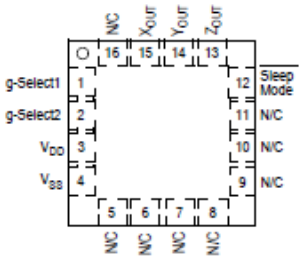


Figure 1. Pin Connections

ORDERING INFORMATION			
Device Name	**Temperture Range**	**Case No.**	**Package**
MMA7260Q	– 20 to +85°C	1622-01	QFN-16, Tube
MMA7260QR2	– 20 to +85°C	1622-01	QFN-16,Tape & Reel

© Freescale Semiconductor, Inc., 2005. All rights reserved.

freescale semiconductor

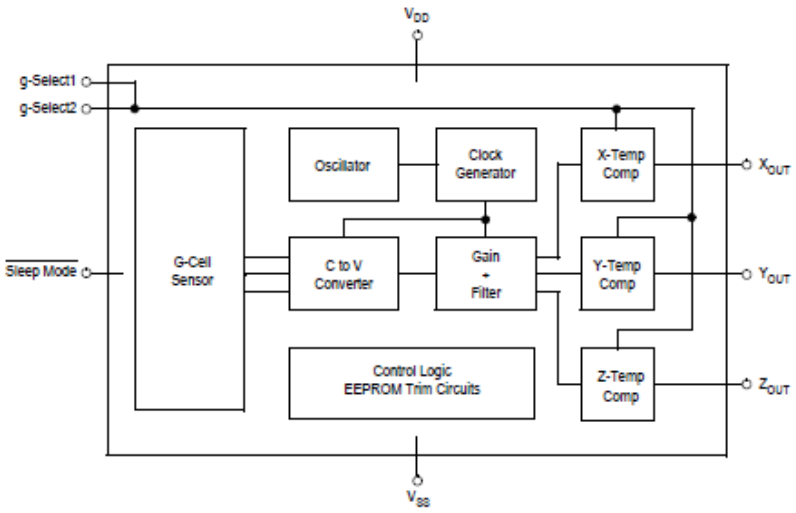


Figure 2. Simplified Accelerometer Functional Block Diagram

Table 1. Maximum Ratings
(Maximum ratings are the limits to which the device can be exposed without causing permanent damage.)

Rating	Symbol	Value	Unit
Maximum Acceleration (all axis)	g_{max}	±2000	g
Supply Voltage	V_{DD}	–0.3 to +3.6	V
Drop Test[1]	D_{drop}	1.8	m
Storage Temperature Range	T_{stg}	–40 to +125	°C

1. Dropped onto concrete surface from any axis.

ELECTRO STATIC DISCHARGE (ESD)

WARNING: This device is sensitive to electrostatic discharge.

Although the Freescale accelerometer contains internal 2000 V ESD protection circuitry, extra precaution must be taken by the user to protect the chip from ESD. A charge of over 2000 volts can accumulate on the human body or associated test equipment. A charge of this magnitude can alter the performance or cause failure of the chip. When handling the accelerometer, proper ESD precautions should be followed to avoid exposing the device to discharges which may be detrimental to its performance.

Table 2. Operating Characteristics

Unless otherwise noted: $-20°C \le T_A \le 85°C$, $2.2\ V \le V_{DD} \le 3.6\ V$, Acceleration = 0g, Loaded output[1]

Characteristic	Symbol	Min	Typ	Max	Unit
Operating Range[2]					
Supply Voltage[3]	V_{DD}	2.2	3.3	3.6	V
Supply Current	I_{DD}	—	500	800	µA
Supply Current at Sleep Mode[4]	I_{DD}	—	3	10	µA
Operating Temperature Range	T_A	–20	—	+85	°C
Acceleration Range, X-Axis, Y-Axis, Z-Axis					
g-Select1 & 2: 00	g_{FS}	—	±1.5	—	g
g-Select1 & 2: 10	g_{FS}	—	±2.0	—	g
g-Select1 & 2: 01	g_{FS}	—	±4.0	—	g
g-Select1 & 2: 11	g_{FS}	—	±6.0	—	g
Output Signal					
Zero g (T_A = 25°C, V_{DD} = 3.3 V)[5]	V_{OFF}	1.485	1.65	1.815	V
Zero g	V_{OFF}, T_A	—	±2	—	mg/°C
Sensitivity (T_A = 25°C, V_{DD} = 3.3 V)					
1.5g	$S_{1.5g}$	740	800	860	mV/g
2g	S_{2g}	555	600	645	mV/g
4g	S_{4g}	277.5	300	322.5	mV/g
6g	S_{6g}	185	200	215	mV/g
Sensitivity	S,T_A	—	±3	—	%/°C
Bandwidth Response					
XY	f_{-3dB}	—	350	—	Hz
Z	f_{-3dB}	—	150	—	Hz
Noise					
RMS (0.1 Hz – 1 kHz)[4]	n_{RMS}	—	4.7	—	mVrms
Power Spectral Density RMS (0.1 Hz – 1 kHz)[4]	n_{PSD}	—	350	—	$\mu g/\sqrt{Hz}$
Control Timing					
Power-Up Response Time[6]	$t_{RESPONSE}$	—	1.0	2.0	ms
Enable Response Time[7]	t_{ENABLE}	—	0.5	2.0	ms
Sensing Element Resonant Frequency					
XY	f_{GCELL}	—	6.0	—	kHz
Z	f_{GCELL}	—	3.4	—	kHz
Internal Sampling Frequency	f_{CLK}	—	11	—	kHz
Output Stage Performance					
Full-Scale Output Range (I_{OUT} = 30 µA)	V_{FSO}	V_{SS}+0.25	—	V_{DD}–0.25	V
Nonlinearity, X_{OUT}, Y_{OUT}, Z_{OUT}	NL_{OUT}	–1.0	—	+1.0	%FSO
Cross-Axis Sensitivity[8]	$V_{XY, XZ, YZ}$	—	—	5.0	%

1. For a loaded output, the measurements are observed after an RC filter consisting of a 1.0 kΩ resistor and a 0.1 µF capacitor to ground.
2. These limits define the range of operation for which the part will meet specification.
3. Within the supply range of 2.2 and 3.6 V, the device operates as a fully calibrated linear accelerometer. Beyond these supply limits the device may operate as a linear device but is not guaranteed to be in calibration.
4. This value is measured with g-Select in 1.5g mode.
5. The device can measure both + and – acceleration. With no input acceleration the output is at midsupply. For positive acceleration the output will increase above $V_{DD}/2$. For negative acceleration, the output will decrease below $V_{DD}/2$.
6. The response time between 10% of full scale Vdd input voltage and 90% of the final operating output voltage.
7. The response time between 10% of full scale Sleep Mode input voltage and 90% of the final operating output voltage.
8. A measure of the device's ability to reject an acceleration applied 90° from the true axis of sensitivity.

PRINCIPLE OF OPERATION

The Freescale accelerometer is a surface-micromachined integrated-circuit accelerometer.

The device consists of two surface micromachined capacitive sensing cells (g-cell) and a signal conditioning ASIC contained in a single integrated circuit package. The sensing elements are sealed hermetically at the wafer level using a bulk micromachined cap wafer.

The g-cell is a mechanical structure formed from semiconductor materials (polysilicon) using semiconductor processes (masking and etching). It can be modeled as a set of beams attached to a movable central mass that move between fixed beams. The movable beams can be deflected from their rest position by subjecting the system to an acceleration (Figure 3) .

As the beams attached to the central mass move, the distance from them to the fixed beams on one side will increase by the same amount that the distance to the fixed beams on the other side decreases. The change in distance is a measure of acceleration.

The g-cell beams form two back-to-back capacitors (Figure 3). As the center beam moves with acceleration, the distance between the beams changes and each capacitor's value will change, ($C = A\varepsilon/D$). Where A is the area of the beam, ε is the dielectric constant, and D is the distance between the beams.

The ASIC uses switched capacitor techniques to measure the g-cell capacitors and extract the acceleration data from the difference between the two capacitors. The ASIC also signal conditions and filters (switched capacitor) the signal, providing a high level output voltage that is ratiometric and proportional to acceleration.

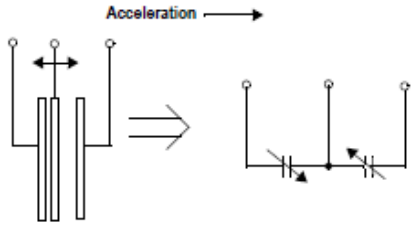


Figure 3. Simplified Transducer Physical Model

SPECIAL FEATURES

g-Select

The g-Select feature allows for the selection among 4 sensitivities present in the device. Depending on the logic input placed on pins 1 and 2, the device internal gain will be changed allowing it to function with a 1.5g, 2g, 4g, or 6g sensitivity (Table 3). This feature is ideal when a product has applications requiring different sensitivities for optimum performance. The sensitivity can be changed at anytime during the operation of the product. The g-Select1 and g-Select2 pins can be left unconnected for applications requiring only a 1.5g sensitivity as the device has an internal pulldown to keep it at that sensitivity (800mV/g).

Table 3. g-Select pin Descriptions

g-Select2	g-Select1	g-Range	Sensitivity
0	0	1.5g	800mV/g
0	1	2g	600mV/g
1	0	4g	300mV/g
1	1	6g	200mV/g

Sleep Mode

The 3 axis accelerometer provides a Sleep Mode that is ideal for battery operated products. When Sleep Mode is active, the device outputs are turned off, providing significant reduction of operating current. A low input signal on pin 12 (Sleep Mode) will place the device in this mode and reduce the current to 3uA typ. For lower power consumption, it is recommended to set g-Select1 and g-Select2 to 1.5g mode. By placing a high input signal on pin 12, the device will resume to normal mode of operation.

Filtering

The 3 axis accelerometer contains onboard single-pole switched capacitor filters. Because the filter is realized using switched capacitor techniques, there is no requirement for external passive components (resistors and capacitors) to set the cut-off frequency.

Ratiometricity

Ratiometricity simply means the output offset voltage and sensitivity will scale linearly with applied supply voltage. That is, as supply voltage is increased, the sensitivity and offset increase linearly; as supply voltage decreases, offset and sensitivity decrease linearly. This is a key feature when interfacing to a microcontroller or an A/D converter because it provides system level cancellation of supply induced errors in the analog to digital conversion process.

BASIC CONNECTIONS

Pin Descriptions

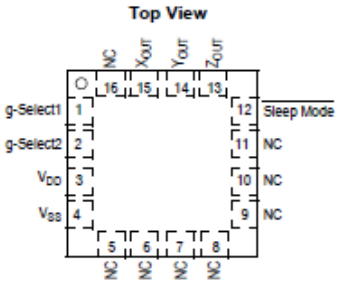


Figure 4. Pinout Description

Table 4. Pin Descriptions

Pin No.	Pin Name	Description
1	g-Select1	Logic input pin to select g level.
2	g-Select2	Logic input pin to select g level.
3	V_{DD}	Power Supply Input
4	V_{SS}	Power Supply Ground
5 - 7	N/C	No internal connection. Leave unconnected.
8 - 11	N/C	Unused for factory trim. Leave unconnected.
12	$\overline{\text{Sleep Mode}}$	Logic input pin to enable product or Sleep Mode.
13	Z_{OUT}	Z direction output voltage.
14	Y_{OUT}	Y direction output voltage.
15	X_{OUT}	X direction output voltage.
16	N/C	No internal connection. Leave unconnected.

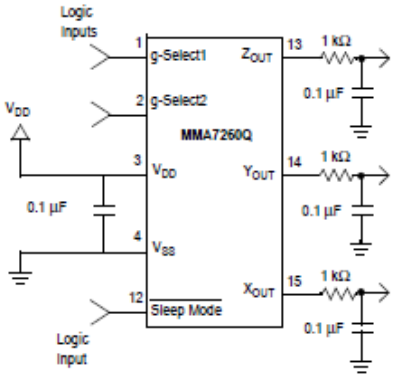


Figure 5. Accelerometer with Recommended Connection Diagram

PCB Layout

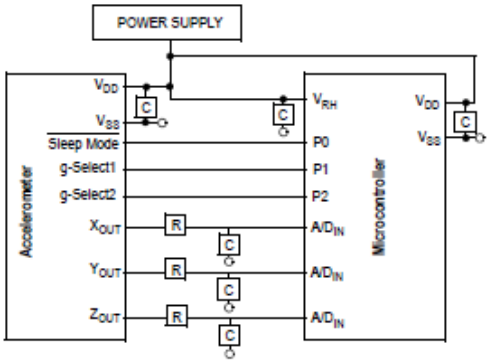
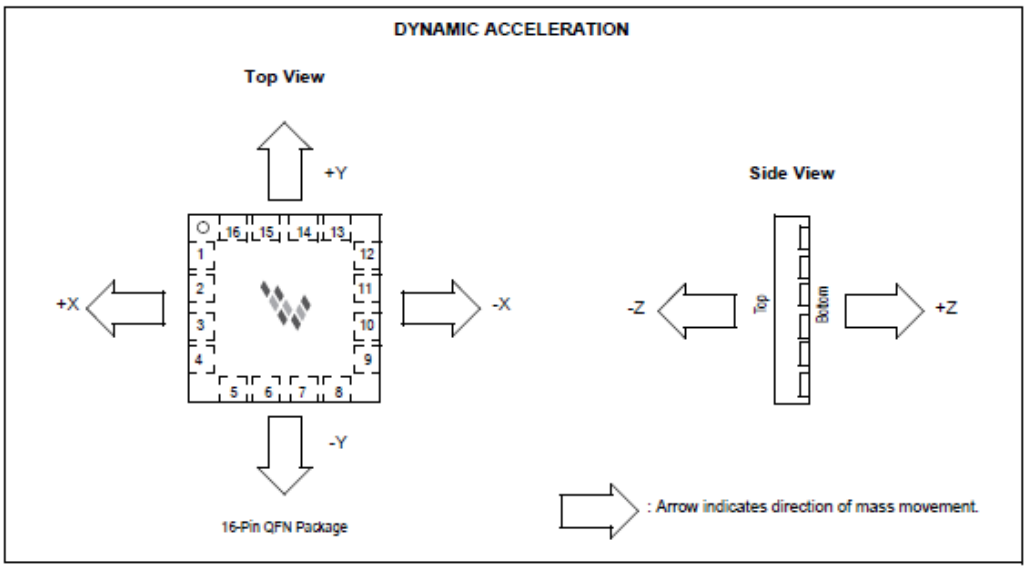
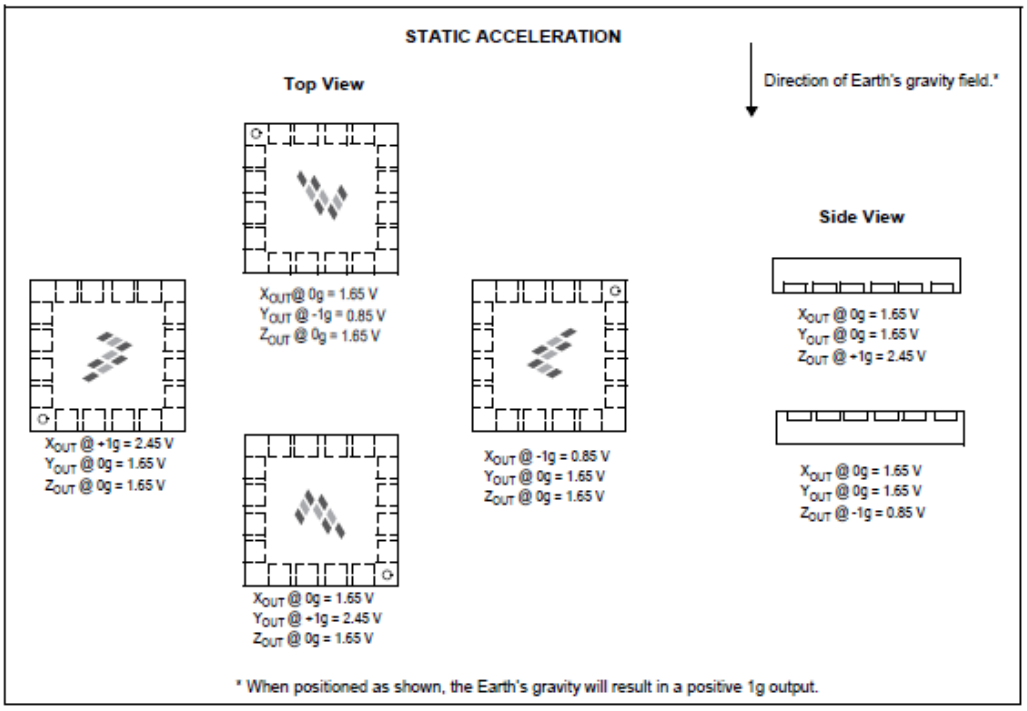


Figure 6Recommended PCB Layout for Interfacing Accelerometer to Microcontroller

NOTES:

1. Use 0.1 µF capacitor on V_{DD} to decouple the power source.
2. Physical coupling distance of the accelerometer to the microcontroller should be minimal.
3. Flag underneath package is connected to ground.
4. Place a ground plane beneath the accelerometer to reduce noise, the ground plane should be attached to all of the open ended terminals shown in Figure 6.
5. Use an RC filter with 1.0 kΩ and 0.1 µF on the outputs of the accelerometer to minimize clock noise (from the switched capacitor filter circuit).
6. PCB layout of power and ground should not couple power supply noise.
7. Accelerometer and microcontroller should not be a high current path.
8. A/D sampling rate and any external power supply switching frequency should be selected such that they do not interfere with the internal accelerometer sampling frequency (11 kHz for the sampling frequency). This will prevent aliasing errors.

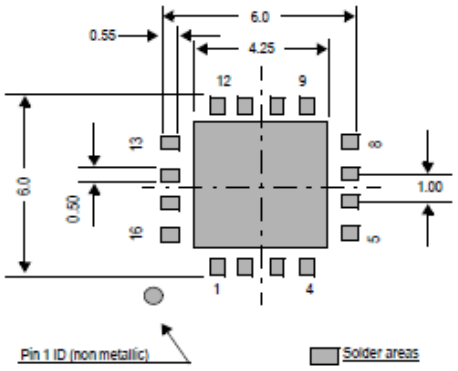

DYNAMIC ACCELERATION
Top View
+Y
+X
-X
-Y
16-Pin QFN Package
Side View
-Z
+Z
Top
Bottom
: Arrow indicates direction of mass movement.


STATIC ACCELERATION
Direction of Earth's gravity field.*
Top View
X_{OUT} @ 0g = 1.65 V
Y_{OUT} @ -1g = 0.85 V
Z_{OUT} @ 0g = 1.65 V
X_{OUT} @ +1g = 2.45 V
Y_{OUT} @ 0g = 1.65 V
Z_{OUT} @ 0g = 1.65 V
X_{OUT} @ -1g = 0.85 V
Y_{OUT} @ 0g = 1.65 V
Z_{OUT} @ 0g = 1.65 V
X_{OUT} @ 0g = 1.65 V
Y_{OUT} @ +1g = 2.45 V
Z_{OUT} @ 0g = 1.65 V
Side View
X_{OUT} @ 0g = 1.65 V
Y_{OUT} @ 0g = 1.65 V
Z_{OUT} @ +1g = 2.45 V
X_{OUT} @ 0g = 1.65 V
Y_{OUT} @ 0g = 1.65 V
Z_{OUT} @ -1g = 0.85 V
* When positioned as shown, the Earth's gravity will result in a positive 1g output.

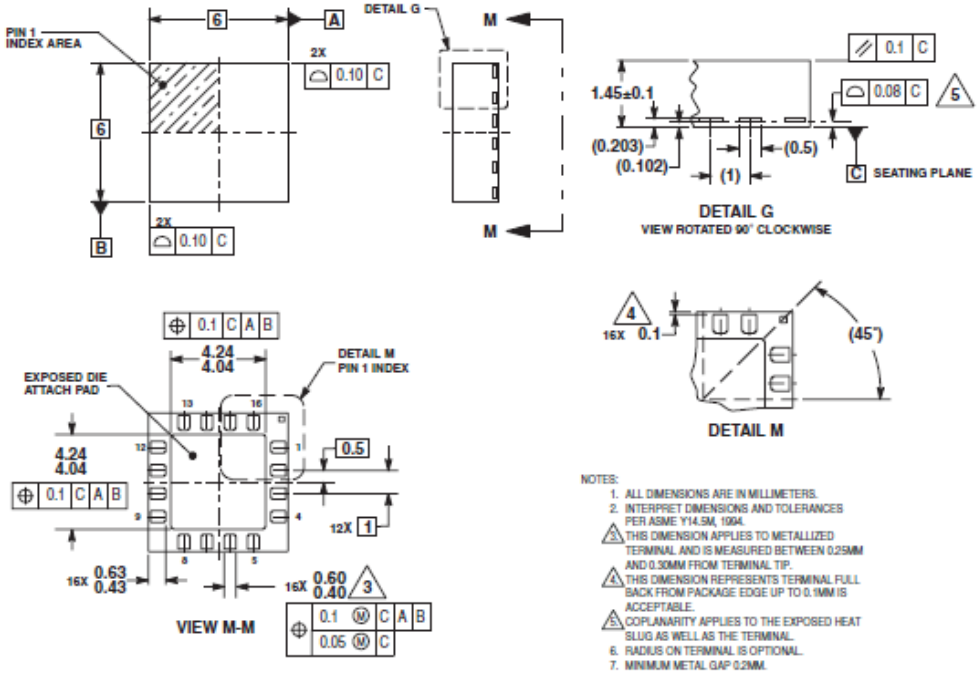
MINIMUM RECOMMENDED FOOTPRINT FOR SURFACE MOUNTED APPLICATIONS

Surface mount board layout is a critical portion of the total design. The footprint for the surface mount packages must be the correct size to ensure proper solder connection interface between the board and the package.

With the correct footprint, the packages will self-align when subjected to a solder reflow process. It is always recommended to design boards with a solder mask layer to avoid bridging and shorting between solder pads.



PACKAGE DIMENSIONS



NOTES:
1. ALL DIMENSIONS ARE IN MILLIMETERS.
2. INTERPRET DIMENSIONS AND TOLERANCES PER ASME Y14.5M, 1994.
3. THIS DIMENSION APPLIES TO METALLIZED TERMINAL AND IS MEASURED BETWEEN 0.25MM AND 0.30MM FROM TERMINAL TIP.
4. THIS DIMENSION REPRESENTS TERMINAL FULL BACK FROM PACKAGE EDGE UP TO 0.1MM IS ACCEPTABLE.
5. COPLANARITY APPLIES TO THE EXPOSED HEAT SLUG AS WELL AS THE TERMINAL.
6. RADIUS ON TERMINAL IS OPTIONAL.
7. MINIMUM METAL GAP 0.2MM.

CASE 1622-01
ISSUE O

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

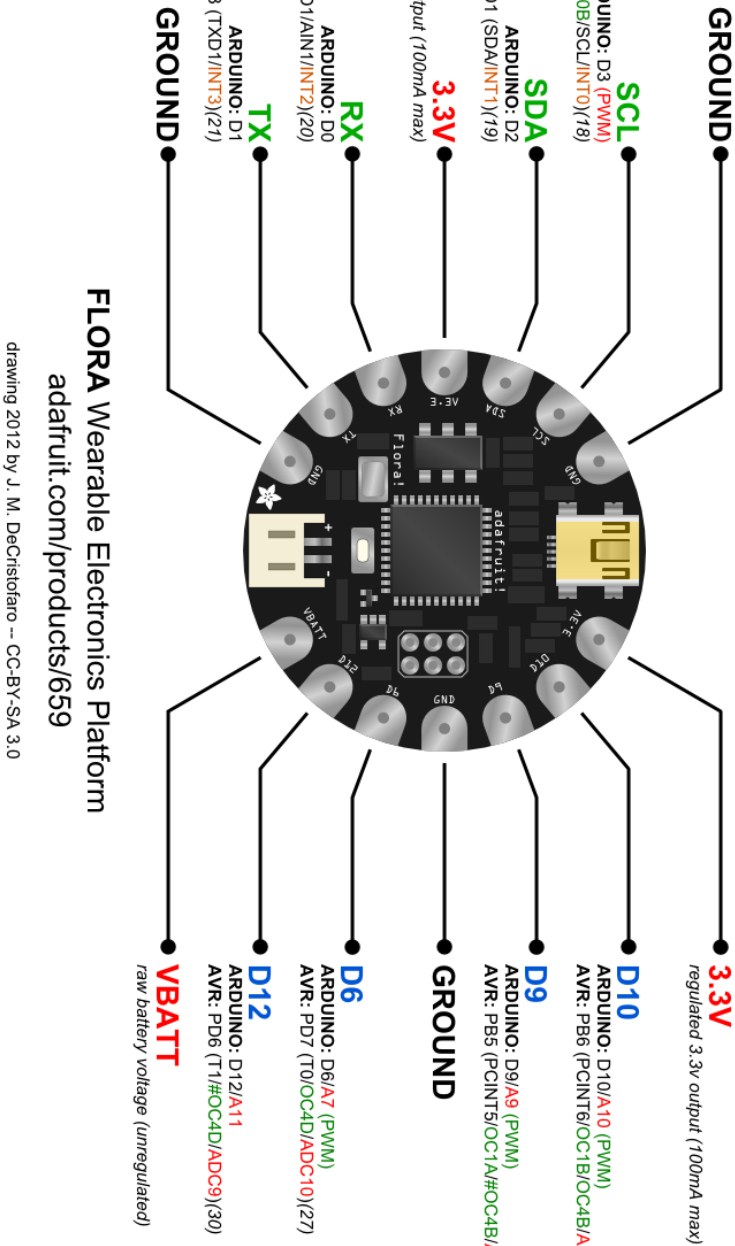
Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.
© Freescale Semiconductor, Inc. 2005. All rights reserved.

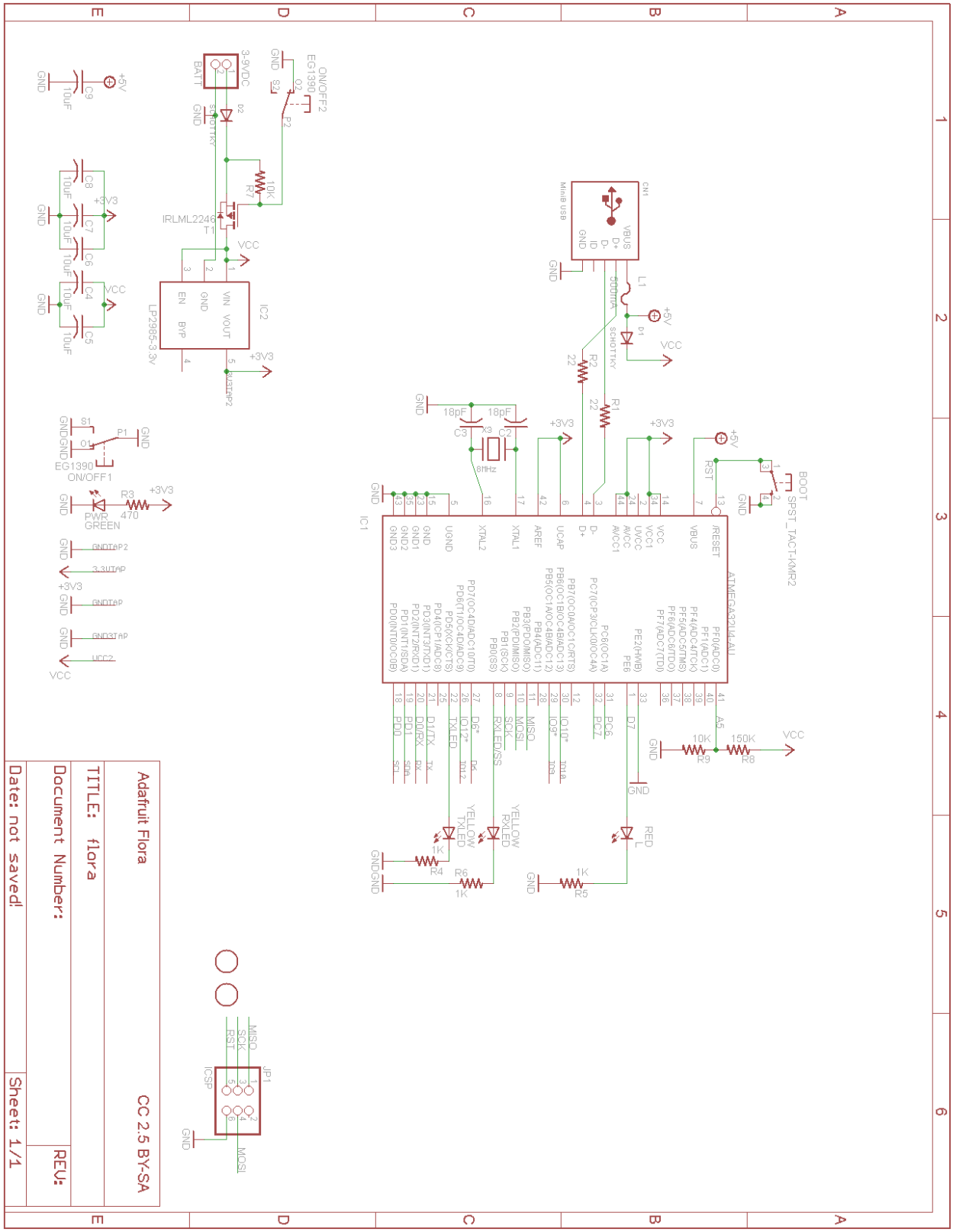
MMA7260Q
Rev. 0
04/2005

C.2 FLORA

C.2.1 Pinout



C.2.2 Esquemático



C.3 Atmel ATmega32U4

Atmel

ATmega16U4/ATmega32U4

8-bit Microcontroller with 16/32K bytes of ISP Flash and USB Controller

DATASHEET

Features

- High Performance, Low Power AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 135 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-Chip 2-cycle Multiplier
- Non-volatile Program and Data Memories
  - 16/32KB of In-System Self-Programmable Flash
  - 1.25/2.5KB Internal SRAM
  - 512Bytes/1KB Internal EEPROM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
    - Parts using external XTAL clock are pre-programed with a default USB bootloader
  - Programming Lock for Software Security
- JTAG (IEEE® std. 1149.1 compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- USB 2.0 Full-speed/Low Speed Device Module with Interrupt on Transfer Completion
  - Complies fully with Universal Serial Bus Specification Rev 2.0
  - Supports data transfer rates up to 12Mbit/s and 1.5Mbit/s
  - Endpoint 0 for Control Transfers: up to 64-bytes
  - Six Programmable Endpoints with IN or Out Directions and with Bulk, Interrupt or Isochronous Transfers
  - Configurable Endpoints size up to 256 bytes in double bank mode
  - Fully independent 832 bytes USB DPRAM for endpoint memory allocation
  - Suspend/Resume Interrupts
  - CPU Reset possible on USB Bus Reset detection
  - 48MHz from PLL for Full-speed Bus Operation
  - USB Bus Connection/Disconnection on Microcontroller Request
  - Crystal-less operation for Low Speed mode
- Peripheral Features
  - On-chip PLL for USB and High Speed Timer: 32 up to 96MHz operation
  - One 8-bit Timer/Counter with Separate Prescaler and Compare Mode

REFERENCIAS

[1] Dirección General de Tráfico, «DGT,» 2013. [En línea].

Available:http://www.dgt.es/Galerias/seguridad-vial/estadisticas-e-indicadores/publicaciones/principales-cifras-siniestralidad/Siniestralidad_Vial_2013.pdf. [Último acceso: Agosto 2016].

[2] Dirección General de Tráfico, «DGT,» 2013. [En línea].

Available:http://www.dgt.es/Galerias/seguridad-vial/estadisticas-e-indicadores/publicaciones/accidentes-urban/Siniestralidad_Urbana_2013_EE.pdf. [Último acceso: Agosto 2016].

[3] Dirección General de Tráfico, «DGT,» [En línea].

Available: https://sedeapl.dgt.gob.es/WEB_IEST_CONSULTA/. [Último acceso: 29 2016 Julio].

[4] Dirección General de Tráfico, «DGT,» 2016. [En línea]. Available: http://www.dgt.es/Galerias/seguridad-vial/educacion-vial/recursos-didacticos/jovenes/Guia-Ciclista-2016.pdf. [Último acceso: Agosto 2016].

[5] Carlijn V. C. Bouten,* Karel T. M. Koekkoek, Maarten Verduin, Rens Kodde, and Jan D. Janssen, «A Triaxial Accelerometer and Portable Data Processing Unit for the Assessment of Daily Physical Activity,» *IEEE TRANSACTIONS ON BIOMEDICAL ENGINEERING,* vol. 44, nº 3, March 1997.

[6] Brian Barkley Graham, «Using an Accelerometer Sensor to Measure Human Hand Motion,» Massachusetts Institute of Technology, 2000.

[7] Alberto Manzanares del Moral, «Estudio de Modelos Matemáticos de Acelerómetros Comerciales,» 2008.

[8] Xu, Yong, «Wearable accelerometers for continuous heart and lung sound monitoring,» Wayne State University, [En línea].

Available: http://www.ece.eng.wayne.edu/~yxu/doc/researches/heart%20and%20lung.htm.

[9] Arduino LilyPad, «LilyPad Website,» [En línea]. Available: http://lilypadarduino.org/?cat=4. [Último acceso: Agosto 2016].

[10] Adafruit, «Adafruit Website,» [En línea]. Available: https://www.adafruit.com/categories/65. [Último acceso: Agosto 2016].

[11] KeKeSmart, «KeKeSmart Website,» [En línea].

Available: http://www.ikeke.co/getting-started/introduction-of-kekepad/. [Último acceso: Septiembre 2016].

[12] Intel, «Intel CUrie,» [En línea]. Available: http://www.intel.com/content/www/us/en/wearables/wearable-soc.html. [Último acceso: Septiembre 2016].

[13] Cyclee, «Cyclee Website,» [En línea]. Available: https://www.behance.net/gallery/26503015/CYCLEE. [Último acceso: Septiembre 2016].

[14] Zackees, «Zackees Website,» [En línea]. Available: https://zackees.com. [Último acceso: Septiembre 2016].

[15] Safe Ride, «Safe Ride Website,» [En línea]. Available: https://fondeadora.mx/projects/safe-ride-mx. [Último acceso: Septiembre 2016].

[16] Lumos, «Lumos Website,» [En línea]. Available: https://lumoshelmet.co. [Último acceso: Septiembre 2016].

[17] Blackpack, «Blackpack Website,» [En línea].

Available: https://www.behance.net/gallery/22026795/Blackpack. [Último acceso: Septiembre 2016].

[18] Stern, Becky, «Getting Started with FLORA,» 2016.

[19] Atmel, «ATmega16U4/ATmega32U4».

[20] Freescale Semiconductor, «MMA7260Q».

[21] Stern, Becky, «Conductive Thread,» 2016.

[22] Brown McFarlane, [En línea]. Available: http://www.brownmac.com/products/stainless-steel-plate/316-and-316l-spanish.aspx. [Último acceso: Agosto 2016].

[23] Emil Martinec - University of Chicago, «Noise, Dynamic Range and Bit Depth in Digital SLRs,» 2008.

[24] «Electrical Engineering Stack Exchange,» [En línea].

Available: http://electronics.stackexchange.com/questions/129325/what-is-the-significance-of-8-bits-in-the-context-of-an-adc0808. [Último acceso: Agosto 2016].

[25] E. W. Weisstein, «Correlation Coefficient,» From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/CorrelationCoefficient.html.

[26] Mathworks, «Mathworks - Documentation,» [En línea].

Available: http://es.mathworks.com/help/images/ref/corr2.html. [Último acceso: Agosto 2016].

[27] «Github,» [En línea]. Available: https://gist.github.com/albertmeronyo/d091fc082c97803f49e9.

[28] «AVR Libc - Standard C library for AVR-GCC,» [En línea]. Available: http://www.nongnu.org/avr-libc/user-manual/group__avr__pgmspace.html. [Último acceso: Agosto 2016].